GEOMETRIC TOOLS, SAMPLING STRATEGIES, BAYESIAN INVERSE PROBLEMS AND DESIGN UNDER UNCERTAINTY

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Henri Raphael Sternfels

January 2013

GEOMETRIC TOOLS, SAMPLING STRATEGIES, BAYESIAN INVERSE PROBLEMS AND DESIGN UNDER UNCERTAINTY

Henri Raphael Sternfels, Ph.D.

Cornell University 2013

The main contributions of the present thesis are novel computational methods related to uncertainty quantification, inverse problems and reduced order modeling in engineering. In the first chapter, we describe a framework to optimize an engineering system under large uncertainties. The optimization problem being recast as a sampling problem, the use of advanced sampling schemes associated with a hierarchical approach using approximate models enables an efficient identification of design values; along with corresponding sensitivity and robustness information. The second chapter deals with the solution of Bayesian inverse problems, in which unknown parameter values in a model are being inferred from uncertain measurements of the output of the system of interest. A reduced order model interpolation scheme, based on differential geometric ideas, enables faster computations during the posterior sampling process while maintaining a high accuracy. Finally, the last chapter proposes a solution to the snapshot selection problem in reduced order modeling, namely how to select the parameters that represent best the system of interest in the parameter space. The approach chosen is to interpret the parameter space as a Riemannian manifold, with a sensitivity related metric emphasizing regions with more information. The numerical applications chosen for each of those problems are engineering oriented, with the corresponding models being discretized using the finite element method.

BIOGRAPHICAL SKETCH

Raphael Sternfels graduated with a *Diplôme d'Ingénieur* from École Centrale Paris, France, in June 2009, following two years in the *Classes préparatoires aux Grandes Écoles*. He joined the M.S./Ph.D. program in Civil and Environmental Engineering at Cornell University in August 2009, and subsequently earned his Master of Science in August 2011.

Dr. Sternfels pursued his doctoral studies within the Structures Group in the School of Civil and Environmental Engineering at Cornell University, with minors in Applied Mathematics and Computational Science and Engineering. While completing his degree requirements, Dr. Sternfels worked as a graduate research assistant and as a teaching assistant in the School of Civil and Environmental Engineering.

Dr. Sternfels has presented his research at major international conference meetings, including the SIAM Annual Meeting (2010, 2012), the U.S. National Congress on Computational Mechanics (2011) and the SIAM Conference on Uncertainty Quantification (2012). He was also selected to participate in the first Sandia Summer Institute (2011) to work on Uncertainty Quantification, at the Livermore site of Sandia National Laboratories.

Dr. Sternfels's dissertation, entitled "Geometric tools, sampling strategies, Bayesian inverse problems and design under uncertainty," was supervised by Prof. Christopher J. Earls.

To my parents, family, and friends.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor Prof. Christopher Earls, who has been much more than an advisor to me. In addition to supervising my thesis research, he has been an invaluable mentor; someone I know I could always ask for advice and not only for scientific concerns. I would also thank the other members of my thesis committee, Prof. Mircea Grigoriu and Prof. Charles Van Loan. On top of appreciating their vast expertise while sitting in their classes, I had the pleasure to have numerous enjoyable conversations with them. I additionally have a special word of thanks for Prof. Phaedon-Stelios Koutsourelakis, now at TU Munich, who provided me guidance during the critical first year of my stay at Cornell. Without him, I wouldn't be at this stage now.

I would also like to thank Prof. Derek Warner, who has been a great TA supervisor. More broadly, I would like to address words of thanks to the entire academic and administrative staff with whom I had the opportunity to interact, and who were part of my enjoyable stay at Cornell. Special words for my fellow CEEGSA officers, it was a pleasure to help making beer widely available to enjoy our Friday nights once in a while.

I want to thank my parents and family, who were able to support me while overseas, and warmly welcome me back home for holidays. Finally, I have a special words for all the friends who made my experience at Cornell, including Alin R., Antoine E., Ashley S., Brett D., Cathy H., Gabriel D., Heather R., Jim W., Jose C., Manuel D., Natalia R., Swarnavo S., Xavier S., and everyone else I forgot to put in this list. Collegetown and further outings, Tuesday night trivia, two Softball intramural championships, etc. were all a great part of my stay in Ithaca.

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Context

Over the past decades, we have witnessed a formidable development in computational tools, as well as a wider availability of computer resources. This enabled engineers to simulate increasingly complex systems.

However, traditional mathematical modeling of engineering and physical systems has long been focused on deterministic formulations, not taking into account uncertainties and their impact on the system's response and overall reliability. Advances in available computational power enabled the development of robust and efficient *uncertainty quantification* computational methods for accurately assessing the consequences of uncertainties; thus leading to improvements in system analysis and design. The first contribution of this thesis deals with the problem of design and control of stochastic systems in the context of random heterogeneous materials.

Additionally, with the ever-increasing complexity of engineering systems, there is a need for rigorous methods for system identification and prognosis in order to enable new understanding of the actual condition and future performance of such systems. The second contribution of this thesis is a computationally efficient probabilistic framework that enables the identification of model parameters from noisy measurements of the response, by means of solving a *Bayesian inverse problem*.

Probabilistic approaches for uncertainty quantification and inverse problems are typically computationally demanding, as several runs of the computer model are needed. The advent of

reduced order modeling tools enabled computationally efficient approximations of the system response, by focusing on salient features obtained from selected runs. The last contribution of this thesis focuses on the selection of such runs so that maximum information about the system response is collected, thus drastically improving accuracy.

Throughout this thesis, there are a few key recurring mathematical and computational concepts, of which we provide a broad overview in the next section.

1.2 Overview of mathematical and computational tools

In this section, we provide a brief overview of the development and goals of a few recurring mathematical and computational concepts.

Finite element methods were developed from the 1960s in order to simulate physical systems modeled by partial differential equations, with applications in solid mechanics, heat transfer, etc. The name "finite element" was first mentioned in [25]. The main idea is to discretize the solution of the governing partial differential equations. To that aim, a weak form of the latter equations is formulated, and the approximate solution is sought in a finite-dimensional space oftentimes resulting from a meshing of the problem domain. In this thesis, computer models of systems being considered in the numerical applications are being developed using the finite element method.

Monte Carlo methods are a set of probabilistic techniques to sample from probabilistic distributions. Of particular interest are Markov chain Monte Carlo (MCMC) and Sequential Monte Carlo (SMC) methods. They enable one to sample from complex probability distribution which cannot be sampled directly. Markov chain Monte Carlo, with roots in [68, 48],

has been recognized as one of the ten most important algorithms of the century. Sequential Monte Carlo methods, of which heavy use is made in this thesis, are general sampling techniques originating from filtering problems [45, 31]. Although such sampling methods started being developed in the 1950s, they were rediscovered in the 1990s by statisticians to support Bayesian computations. Bayesian inference enables one to update in a principled way prior information or beliefs with information from data –which is of particular interest in the context of inverse problems– by means of the celebrated Bayes' theorem.

In the later parts of this thesis, we make use of powerful arguments emanating from Riemannian geometry. Riemannian geometry, named after the German mathematician Bernhard Riemann, is a branch of differential geometry which enables to study curved spaces, which metric properties can vary from point to point. It is also a powerful tool to devise geometrical arguments on spaces that are not vector spaces, but can be found to have a Riemannian structure.

The following chapters will provide more detailed reviews of each of those concepts, as needed during the development of the various contributions of this thesis.

1.3 Outline

The remainder of this thesis is organized as follows: in Chapters 2-3-4, we develop the three main contributions of this thesis mentioned earlier in section 1.1. Those chapters are self-contained and can be read independently. The appendix describes a software contribution developed to support the research efforts leading to this thesis.

CHAPTER 2

STOCHASTIC DESIGN AND CONTROL IN RANDOM HETEROGENEOUS MATERIALS

Abstract

The present chapter discusses a sampling framework that enables the optimization of complex systems characterized by high-dimensional uncertainties and design variables. We are especially concerned with problems relating to random heterogeneous materials where uncertainties arise from the stochastic variability of their properties. In particular, we reformulate topology optimization problems to account for the design of truly random composites. In addition, we address the optimal prescription of input loads/excitations in order to achieve a target response by the random material system. The methodological advances proposed in this work consist of an adaptive Sequential Monte Carlo scheme that economizes the number of runs of the forward solver and allows the analyst to identify several local maxima that provide important information with regards to the robustness of the design. We further propose a principled manner of introducing information from approximate models that can ultimately lead to further reductions in computational cost.

2.1 Introduction

The analysis of materials which exhibit very small length scales of heterogeneity has attracted considerable attention in recent years [106]. This is because fine details in the microstructure can give rise to marked differences in the macroscale response. In reality, the majority of such materials exhibit randomness as local physical and mechanical properties fluctuate

stochastically. In multiphase materials for example the distribution of the constituent phases in space does not follow a particular pattern and is characterized by disorder. It is therefore obvious that a probabilistic description is most appropriate and provides a sounder basis for their characterization and quantification of the performance of the systems where these appear.

While marked advances have been achieved in the context of modeling [111, 60, 59, 89] and uncertainty propagation [103, 22, 110, 107], the problem of design/optimization in the presence of randomness has received much less attention [86, 112]. We address two problems in the context of random heterogeneous materials. On one hand we examine the problem of designing random heterogeneous materials. This can be seen as the fully stochastic counterpart of topology optimization, a deterministic tool for the systematic design of composite microstructures with desirable macroscopic properties [93, 92, 94, 99, 95, 100, 101, 11, 97, 96, 41, 43]. Rather than fully specifying the spatial distribution of the constitutive phases, we are interested in controlling statistics of the associated probability distribution (i.e. volume fractions, spatial correlations), while the resulting microstructure remains random. Such a capability could prove particularly useful in the fabrication of heterogeneous materials. For example in polycrystalline materials, it is known that macro-scale forming parameters such as forging rates, die shapes and preform shapes or heat treatment, do not uniquely define the final polycrystalline texture but rather its statistical features [105]. In naturally occurring random heterogeneous materials such as soils, remediation procedures such as solidification or stabilization allow us to alter the microstructure and its properties by altering the probabilistic characteristics of the existing phases and introducing randomly dispersed new materials.

The second problem of interest involves the optimization of the input in the presence of uncertainties. In particular we consider a random heterogeneous material, fully defined

by the probability distribution of its properties and seek to identify the input such that the *expected* response is as close as possible to a target, desired output [112]. Both problems are examined in the context of heat conduction but can be readily extended to other physics as the methodology advocated makes non-intrusive use of forward solvers (e.g. Finite Element codes).

The present work advocates a simulation strategy that recasts the optimization problem as a sampling problem in the expanded space which apart from the random variables includes the design/control parameters [70]. High probability regions of the auxiliary target density correspond to maxima of the design objectives. To that end we employ adaptive *Sequential Monte Carlo* (SMC) methods that are well-suited in high dimensions, and are directly parallelizable and capable of identifying multimodal densities which correspond to local maxima [28, 61]. In addition, they give rise to a population of estimates that provide valuable information with regards to the robustness of the optima. Despite their advantages, SMC-based samplers require multiple calls to deterministic simulators which might be impractical or infeasible for highly complex, nonlinear solvers. To that end we propose an adaptive sampling framework which aims at reducing the number of calls to the forward solver in order to attain a certain level of accuracy. Furthermore, we propose a hierarchical strategy where *approximate* forward models can be rigorously incorporated in order to give accurate estimates at a reduced computational cost. Such approximate solvers can be derived by using principled reduced-order approximations or simply by coarsening the discretization (e.g. in Finite Element analysis).

2.2 Proposed approach

We consider systems characterized by a vector of uncertainties denoted by $\boldsymbol{\theta} \in \boldsymbol{\Theta} \subset \mathbb{R}^{n_\theta}$ distributed according to the density $p(\boldsymbol{\theta})$. This vector will characterize the spatial variability of material properties (e.g. conductivity at various locations) and in general its dimension will be large (i.e. $n_\theta \gg 1$). We denote by $\boldsymbol{d} \in \mathcal{D} \subset \mathbb{R}^{N_d}$ the vector of design/control variables. In the problems examined in this chapter, these represent the input or the statistics of the distribution of $\boldsymbol{\theta}$, i.e. $p(\boldsymbol{\theta}|\boldsymbol{d})$. Our goal is to find the values of the design variables $\boldsymbol{d}$ in the feasible domain $\mathcal{D}$, that maximize the *expected utility* $\hat{U}(\boldsymbol{d})$:

$$\hat{U}(\boldsymbol{d}) = \int_\Theta U(\boldsymbol{\theta}, \boldsymbol{d}) p(\boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{\theta}. \tag{2.1}$$

The utility function $U(\boldsymbol{\theta}, \boldsymbol{d})$ depends on the output of interest (and is therefore a function of the system uncertainties and design variables) and in the present framework it suffices that it is non-negative $\forall \boldsymbol{\theta}, \boldsymbol{d}$ [1]. The formulation above is quite general and can readily be adapted to cases of practical interest. For example if $U(\boldsymbol{\theta}, \boldsymbol{d}) = \mathbf{1}_{\mathcal{A}}(\boldsymbol{\theta}, \boldsymbol{d})$ is the indicator function of an event $\mathcal{A}$ of interest (e.g. failure, or exceedance of a response threshold) then maximizing $\hat{U}(\boldsymbol{d})$ in Equation (2.1) is equivalent to the maximization of the probability associated with the event $\mathcal{A}$ (similarly one can minimize the probability of event $\mathcal{A}$ by employing the indicator function of the complementary even $\mathcal{A}^c$ in place of U in Equation (2.1)). Furthermore if $\boldsymbol{u}(\boldsymbol{\theta}, \boldsymbol{d})$ denotes the output vector (i.e. displacements, stresses, temperatures etc) and $\boldsymbol{u}_{target}$ a target/desired response, then using $U(\boldsymbol{\theta}, \boldsymbol{d}) = \exp\left(-\|\boldsymbol{u}(\boldsymbol{\theta}, \boldsymbol{d}) - \boldsymbol{u}_{target}\|\right)$ in Equation (2.1) implies finding $\boldsymbol{d}$ for which the response is, on average, as close to the target.

The systems of interest are considered complex in the sense that the utility function $U(\boldsymbol{\theta}, \boldsymbol{d})$ is not known explicitly, and can only be computed for a particular $(\boldsymbol{\theta}, \boldsymbol{d})$ with a call to a, potentially costly, deterministic solver, e.g. a finite element solver. Thus the

[1] Even if U is negative, it suffices that $U(\boldsymbol{\theta}, \boldsymbol{d}) \geq U_0 > -\infty$ in which case one can utilize $U(\boldsymbol{\theta}, \boldsymbol{d}) - U_0 \geq 0$.

number of calls to such a solver will dominate the total amount of computational work. It is clear that a brute force application of deterministic optimization procedures directly on $\hat{U}(\boldsymbol{d})$ is impractical, as each evaluation of this function (and potentially its derivatives) will in general require a high-dimensional integration with respect to the uncertainties $\boldsymbol{\theta}$. Furthermore, discretizing the design space $\mathcal{D}$ is inefficient or infeasible when the dimension n_d of $\boldsymbol{d}$ is large.

For these reasons we advocate a *sampling* strategy, first proposed in [70]. This entails defining an (unormalized) probability density $\pi(\boldsymbol{\theta}, \boldsymbol{d})$ defined on the joint space $\boldsymbol{\Theta} \times \mathcal{D}$:

$$\pi(\boldsymbol{\theta}, \boldsymbol{d}) \propto U(\boldsymbol{\theta}, \boldsymbol{d}) p(\boldsymbol{\theta})\ 1_{\mathcal{D}}(\boldsymbol{d}) \tag{2.2}$$

where $1_{\mathcal{D}}(\boldsymbol{d})$ is the indicator function of the feasible domain $\mathcal{D}$. The marginal $\pi(\boldsymbol{d}) = \int \pi(\boldsymbol{\theta}, \boldsymbol{d})\, \mathrm{d}\boldsymbol{\theta}$ is clearly proportional to $\hat{U}(\boldsymbol{d})$ and as a result samples drawn from the joint density $\pi(\boldsymbol{\theta}, \boldsymbol{d})$ will be marginally distributed according to $\hat{U}(\boldsymbol{d})$ and populate regions where this attains its maximum value(s). It is also important to point out that such an approach does not lead to point estimates for the maxima of the expected utility but also provides information about the variability of the latter with respect to $\boldsymbol{d}$ and therefore the *robustness* of the select design $\boldsymbol{d}$ [90, 98]. If the global maximum of $\hat{U}(\boldsymbol{d})$ is desired, then this can be achieved within the same framework by artificially expanding the sampling space [33, 53]. In particular, one employs the (unormalized) density $\pi(\boldsymbol{\theta}_{1:n}, \boldsymbol{d})$, where $\boldsymbol{\theta}_{1:n} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_n)$, which is defined on $\underbrace{\boldsymbol{\Theta} \times \ldots \boldsymbol{\Theta}}_{n\ times} \times \mathcal{D}$ [70]:

$$\pi(\boldsymbol{\theta}_{1:n}, \boldsymbol{d}) \propto \prod_{j=1}^{n} U(\boldsymbol{\theta}_i, \boldsymbol{d}) p(\boldsymbol{\theta}_i)\ 1_{\mathcal{D}}(\boldsymbol{d}). \tag{2.3}$$

It is clear again that the marginal with respect to $\boldsymbol{d}$, i.e. $\pi(\boldsymbol{d}) = \int \pi(\boldsymbol{\theta}_{1:n}, \boldsymbol{d})\, \mathrm{d}\boldsymbol{\theta}_{1:n}$ is proportional to $\hat{U}^n(\boldsymbol{d})$. As a result, the $\boldsymbol{d}$-coordinates of samples drawn from $\pi(\boldsymbol{\theta}_{1:n}, \boldsymbol{d})$ will be more tightly concentrated around the maxima of $\hat{U}(\boldsymbol{d})$, increasingly so with n.

Despite its flexibility, such an approach requires sampling in the *joint* space of random and design variables. Its dimension is even higher when the augmented density of Equation (2.3) is employed. While Monte Carlo strategies offer the most general method for carrying out the sampling task, a naive implementation would be impractical as it would require a large number of evaluations of the utility function $U(\boldsymbol{\theta}, \boldsymbol{d})$ and therefore a lot of calls to the forward solver. Furthemore, it might fail to identify all the modes of the distribution $\pi(\boldsymbol{\theta}, \boldsymbol{d})$ in Equation (2.2) (or $\pi(\boldsymbol{\theta}_{1:n}, \boldsymbol{d})$ in Equation (2.3)) which correspond to local maxima of the expected utility $\hat{U}(\boldsymbol{d})$. These local maxima can be of considerable value in terms of their physical and engineering significance as they also reveal valuable features with regards to the sensitivity of the expected output to the design/control variables. Traditionally *Markov Chain Monte Carlo* techniques (MCMC) have been employed which are based on building a Markov chain that asymptotically converges to the target density (in this case π) by appropriately defining a transition kernel. While convergence can be assured under weak conditions [63, 83], the rate of convergence can be extremely slow and require a lot of utility function evaluations. Particularly in cases where the target density has multiple modes, very large *mixing times* might be required. For that purpose we advocate the use of Sequential Monte Carlo (SMC) procedures which have the capability of sampling from multi-modal distributions in high-dimensional spaces and discuss an *adaptive* scheme for reducing the computational cost. It is noted that SMC strategies have been employed in this framework and in the context of Bayesian optimal design in [7, 61] and for maximum likelihood estimation in latent variable models in [53].

2.2.1 Adaptive Sequential Monte Carlo

SMC strategies represent a set of flexible simulation-based methods for sampling from a *sequence* of probability distributions [65, 32, 31]. As with Markov Chain Monte Carlo methods (MCMC, [83, 68, 48]), the target distribution(s) need only be known up to a constant as is the case in Equation (2.2) and Equation (2.3). They utilize a set of random samples (commonly referred to as *particles*), which are propagated using a combination of *importance sampling*, *resampling* and MCMC-based *rejuvenation* mechanisms [28, 29]. Each of these particles is associated with an *importance weight*. These weights are updated sequentially along with the particle locations. Hence if $\{(\boldsymbol{\theta}^{(i)}, \boldsymbol{d}^{(i)}),\ w^{(i)}\}_{i=1}^{N}$ represent N such particles and associated weights for distribution $\pi(\boldsymbol{\theta}, \boldsymbol{d})$ in Equation (2.2) then:

$$\pi(\boldsymbol{\theta}, \boldsymbol{d}) \approx \sum_{i=1}^{N} W^{(i)}\ \delta_{\boldsymbol{\theta}^{(i)}}(\boldsymbol{\theta})\ \delta_{\boldsymbol{d}^{(i)}}(\boldsymbol{d}) \tag{2.4}$$

where $W^{(i)} = w^{(i)} / \sum_{k=1}^{N} w^{(k)}$ are the normalized weights and $\delta_{\boldsymbol{x}}(.)$ is the Dirac delta function centered at $\boldsymbol{x}$. Furthermore, for any function $h(\boldsymbol{\theta}, \boldsymbol{d})$ which is π-integrable [27, 24]:

$$\sum_{i=1}^{N} W^{(i)}\ h(\boldsymbol{\theta}^{(i)}, \boldsymbol{d}^{(i)}) \rightarrow \int h(\boldsymbol{\theta}, \boldsymbol{d})\ \pi(\boldsymbol{\theta}, \boldsymbol{d})\ \mathrm{d}\boldsymbol{\theta}\mathrm{d}\boldsymbol{d}. \quad \text{(almost surely)} \tag{2.5}$$

The underlying idea in SMC strategies is to operate on a sequence of distributions, starting from one that can be accurately and easily sampled from, and gradually changing it until the target density is reached. In that respect, it is quite similar to simulated annealing employed in optimization but more general sequences of distributions can be adopted as it will be demonstrated in subsequent sections. However we initially consider a sequence parameterized by $\gamma \in [0, 1]$ which plays the role of reciprocal temperature such that:

$$\pi_{\gamma}(\boldsymbol{\theta}, \boldsymbol{d}) \propto U^{\gamma}(\boldsymbol{\theta}, \boldsymbol{d}) p(\boldsymbol{\theta})\ 1_{\mathcal{D}}(\boldsymbol{d}). \tag{2.6}$$

It is easily seen that for $\gamma = 0$ one recovers the distribution of $p(\boldsymbol{\theta})\ 1_{\mathcal{D}}(\boldsymbol{d})$ i.e. the random variables $\boldsymbol{\theta}$ is distributed according to their density $p(\boldsymbol{\theta})$ and the design variables $\boldsymbol{d}$ uniformly

in the feasible domain $\mathcal{D}$. It is implicitly assumed that generating samples from these distributions is tractable. For $\gamma = 1$, one recovers the target density of Equation (2.2). Starting with a particulate approximation for $\pi_{\gamma=0}$ (which trivially involves drawing samples from the $p(\boldsymbol{\theta})$ for $\boldsymbol{\theta}$ and the uniform in $\mathcal{D}$ for $\boldsymbol{d}$ with weights $w^{(i)} = 1$), the goal is to gradually update the importance weights and particle locations in order to approximate the target density. The role of these auxiliary distributions π_γ for $\gamma \in (0, 1)$ is to *bridge the gap between* $\pi_{\gamma=0}$ *and* $\pi_{\gamma=1}$ and provide a smooth transition path where importance sampling can be efficiently applied. It is easily understood that as the number of these distributions increases the accuracy increases since the transition becomes smoother, but at the same time so does the computational cost as more evaluations of the utility function would be needed. On the other hand too few intermediate distributions π_γ can adversely affect the overall accuracy of the approximation.

To that end we propose an *adaptive* SMC scheme that automatically determines the number of intermediate distributions needed [28, 57]. In this process we are guided by the Effective Sample Size (ESS, [63]). In particular, let S be the total number of intermediate distributions (which is unknown a priori) and $\gamma_s,\ s = 1, 2, \ldots, S$ the associated reciprocal temperatures such that $0 = \gamma_1 < \gamma_2 < \ldots < \gamma_S = 1$, which are also unknown a priori. Let also $\{(\boldsymbol{\theta}_s^{(i)}, \boldsymbol{d}_s^{(i)}),\ W_s^{(i)}\}_{i=1}^N$ denote the particulate approximation of π_{γ_s} defined as in Equation (2.6) for $\gamma = \gamma_s$. The Effective Sample Size of these particles is then defined as $ESS_s = 1/\sum_{i=1}^N (W_s^{(i)})^2$ and provides a measure of the population variance. One extreme, i.e. when $ESS_s = 1$, arises when a single particle has a unit normalized weight whereas the rest have zero weights and as a result provide no information. The other extreme, i.e. $ESS_s = N$, arises when all the particles are equally informative and have equal weights $W_s^{(i)} = 1/N$.

If the next bridging distribution $\pi_{\gamma_{s+1}}$ is very similar to π_{γ_s} (ie. $\gamma_{s+1} \approx \gamma_s$), then ESS_{s+1}

should not be that much different from ESS_s. On the other hand if that difference is pronounced then ESS_{s+1} could drop dramatically. Hence in determining the next auxiliary distribution, we define an acceptable reduction in the ESS, i.e. $ESS_{s+1} \geqslant \zeta\ ESS_s$ (where $\zeta < 1$) and prescribe γ_{s+1} (Equation (2.6)) accordingly. The proposed adaptive SMC algorithm is summarized in Algorithm 1.

Algorithm 1: Adaptive SMC algorithm

Initialization: Set $s = 1$ and $\gamma_1 = 0$. Initialize population $\{(\boldsymbol{\theta}_1^{(i)}, \boldsymbol{d}_1^{(i)}),\ w_1^{(i)}\}_{i=1}^N$ where $\boldsymbol{\theta}_1^{(i)}$ are i.i.d draws from $p(\boldsymbol{\theta})$, $\boldsymbol{d}^{(i)}$ are i.i.d draws from $1_{\mathcal{D}}(\boldsymbol{d})$ and $w_1^{(i)} = 1$ ($ESS_1 = N$).

while $\gamma_s < 1$ **do**

Set $s = s + 1$.

Reweighing-Importance Sampling: If:

$$w_s^{(i)}(\gamma_s) = w_{s-1}^{(i)}\ \frac{\pi_{\gamma_s}(\boldsymbol{\theta}_{s-1}^{(i)}, \boldsymbol{d}_{s-1}^{(i)})}{\pi_{\gamma_{s-1}}(\boldsymbol{\theta}_{s-1}^{(i)}, \boldsymbol{d}_{s-1}^{(i)})} = w_{s-1}^{(i)}\ U^{\gamma_s - \gamma_{s-1}}(\boldsymbol{\theta}_{s-1}^{(i)}, \boldsymbol{d}_{s-1}^{(i)}) \tag{2.7}$$

are the *updated* weights as a function of γ_s, then determine $\gamma_s \in (\gamma_{s-1},\ 1]$ so that the associated $ESS_s = \zeta ESS_{s-1}$ (the value $\zeta = 0.95$ was used in all the examples). Calculate $w_s^{(i)}$ for this γ_s.

Resampling: If $ESS_s \leqslant ESS_{min}$ then resample (the value $ESS_{min} = N/2$ was used in all the examples).

Rejuvenation: Use a MCMC kernel $P_s\left((\boldsymbol{\theta}_{s-1}^{(i)}, \boldsymbol{d}_{s-1}^{(i)}), (\boldsymbol{\theta}_s^{(i)}, \boldsymbol{d}_s^{(i)})\right)$ that leaves π_{γ_s} invariant to perturb each particle $(\boldsymbol{\theta}_{s-1}^{(i)}, \boldsymbol{d}_{s-1}^{(i)}) \to (\boldsymbol{\theta}_s^{(i)}, \boldsymbol{d}_s^{(i)})$.

The current population $\{(\boldsymbol{\theta}_s^{(i)}, \boldsymbol{d}_s^{(i)}), w_s^{(i)}\}_{i=1}^N$ provides a particulate approximation of π_{γ_s} in the sense of Equations (2.4), (2.5).

end while

It should be noted that unlike MCMC schemes, the particle perturbations in the *Rejuve-*

nation step do not require that the $P_s(.,.)$ is *ergodic* [28]. It suffices that it is a π_{γ_s}-invariant kernel, which readily allows adaptively changing its parameters in order to achieve better mixing rates. A more detailed discussion on the kernels and adaptivity schemes used in the Rejuvenation step is deferred for section 2.3.

The same idea can be employed in sampling in the extended space with respect to the density $\pi(\boldsymbol{\theta}_{1:n}, \boldsymbol{d}),\ n = 1, 2, \ldots$ in Equation (2.3). Specifically, suppose $\{(\boldsymbol{\theta}_{1:n-1}^{(i)}, \boldsymbol{d}^{(i)}), W^{(i)}\}_{i=1}^{N}$ represents a particulate approximation of the density $\pi(\boldsymbol{\theta}_{1:n-1}, \boldsymbol{d})$ defined in Equation (2.3). In order to obtain samples from $\pi(\boldsymbol{\theta}_{1:n}, \boldsymbol{d})$ a new sequence of bridging distributions is built in the spirit of Equation (2.6) as follows:

$$\pi_{n-1,\gamma} \propto U^{\gamma}(\boldsymbol{\theta}_n, \boldsymbol{d}) p(\boldsymbol{\theta}_n) \prod_{j=1}^{n-1} U(\boldsymbol{\theta}_i, \boldsymbol{d}) p(\boldsymbol{\theta}_i)\ 1_{\mathcal{D}}(\boldsymbol{d}), \quad \gamma \in [0, 1]. \tag{2.8}$$

It is immediately obvious that for $\gamma = 0$ we recover $\pi(\boldsymbol{\theta}_{1:n-1}, \boldsymbol{d})$ and for $\gamma = 1$ the target $\pi(\boldsymbol{\theta}_{1:n}, \boldsymbol{d})$. The adaptive SMC scheme of Algorithm 1 can be applied identically for the aforementioned sequence. An additional advantage of the proposed approach is that the state augmentation takes place sequentially which gives the analyst the opportunity to terminate the algorithm if sufficient information on the maximum (or maxima) of the expected utility $\hat{U}(\boldsymbol{d})$ has been obtained.

2.2.2 Using information from approximate models

The proposed adaptive SMC sampling framework allows for great flexibility in selecting the sequence of distributions which can be adapted to the specifics of the problem. In the following we present an alternative that can lead to significant computational savings.

It is clear that for cases of practical interest the most expensive part of the computations relates to the repeated evaluations of the utility function $U(\boldsymbol{\theta}, \boldsymbol{d})$ and in particular, in the

Rejuvenation step of the algorithm in Table 1 (the utility functions values can be stored in memory and used in the Reweighing step). This is because each evaluation implies a run of the forward solver, which can in many cases be costly. For that purpose we propose employing *approximate* computational models, which might be less expensive but provide inaccurate evaluations of the utility function $U(\boldsymbol{\theta}, \boldsymbol{d})$. This idea has been successfully employed in [56, 57].

Such inexpensive, approximate models can be formally constructed using reduced-order modeling strategies [18], or less-rigorously by coarsening the spatio-temporal discretization of the governing PDEs or increasing the allowable error if iterative solvers are used for the solution of the system of algebraic equations. As it has also been demonstrated in [57], it is not important that the solutions of the approximate solver deviate significantly from the reference, but it suffices that they exhibit some sort of dependence in the sense to be explained in the sequence. For clarity of the presentation we assume that the approximate model consists of a coarsened discretization of the governing PDEs, and the utility function evaluated by this solver is denoted by $U_c(\boldsymbol{\theta}, \boldsymbol{d})$. In contrast, the utility function of the reference/fine solver with the desired discretization is denoted by $U_f(\boldsymbol{\theta}, \boldsymbol{d})$. The goal is to sample from $\pi_f(\boldsymbol{\theta}, \boldsymbol{d}) \propto U_f(\boldsymbol{\theta}, \boldsymbol{d}) p(\boldsymbol{\theta}) 1_{\mathcal{D}}(\boldsymbol{d})$ as in Equation (2.2). An important observation to be made is that only the last distribution of the sequence has to depend explicitly on the most accurate utility function. The ideas presented can be readily generalized to a sequence of approximate models with increasing resolution, something that might be desired in practical cases where it is not a priori known what the appropriate resolution should be.

In order to make use of the information furnished by the approximate solver, we first define a sequence of distributions which employ $U_c(\boldsymbol{\theta}, \boldsymbol{d})$ as follows:

$$\pi_{\hat{\gamma}}(\boldsymbol{\theta}, \boldsymbol{d}) \propto U_c^{\hat{\gamma}}(\boldsymbol{\theta}, \boldsymbol{d}) p(\boldsymbol{\theta}) 1_{\mathcal{D}}(\boldsymbol{d}), \quad \hat{\gamma} \in [0, 1]. \tag{2.9}$$

The target density which is obtained for $\hat{\gamma} = 1$ is $\pi_c(\boldsymbol{\theta}, \boldsymbol{d}) \propto U_c(\boldsymbol{\theta}, \boldsymbol{d}) p(\boldsymbol{\theta}) 1_{\mathcal{D}}(\boldsymbol{d})$. Sampling from this sequence of distribution can be achieved using the adaptive SMC scheme of Table 1. It is important to note that due the reduced cost associated which each evaluation of U_c the overall expense is generally much lower than trying to sample from π_f. Once a particulate approximation of π_c has been obtained, we propose operating on the following sequence:

$$\pi_{\tilde{\gamma}}(\boldsymbol{\theta}, \boldsymbol{d}) \propto U_c^{1-\tilde{\gamma}}(\boldsymbol{\theta}, \boldsymbol{d}) U_f^{\tilde{\gamma}}(\boldsymbol{\theta}, \boldsymbol{d})\ p(\boldsymbol{\theta}) 1_{\mathcal{D}}(\boldsymbol{d}), \quad \tilde{\gamma} \in [0, 1] \tag{2.10}$$

which for $\tilde{\gamma} = 1$ recovers the target density $\pi_f(\boldsymbol{\theta}, \boldsymbol{d})$. The adaptive SMC scheme of Table 1 can be readily with a slight change in the Reweighing step where the updated weights of Equation (2.7) are now given by:

$$w_s^{(i)}(\tilde{\gamma}_s) = w_{s-1}^{(i)} \frac{\pi_{\tilde{\gamma}_s}(\boldsymbol{\theta}_{s-1}^{(i)}, \boldsymbol{d}_{s-1}^{(i)})}{\pi_{\tilde{\gamma}_{s-1}}(\boldsymbol{\theta}_{s-1}^{(i)}, \boldsymbol{d}_{s-1}^{(i)})} = w_{s-1}^{(i)} \left(\frac{U_f(\boldsymbol{\theta}_{s-1}^{(i)}, \boldsymbol{d}_{s-1}^{(i)})}{U_c(\boldsymbol{\theta}_{s-1}^{(i)}, \boldsymbol{d}_{s-1}^{(i)})} \right)^{\Delta\tilde{\gamma}_s} \tag{2.11}$$

where $\Delta\tilde{\gamma}_s = \tilde{\gamma}_s - \tilde{\gamma}_{s-1}$. This reweighing has to be performed for all intermediate steps $s = 1, \ldots, S$ and since evaluations of the expensive utility U_f are required at each iteration (and each particle), the overall cost is proportional to S. The expression above implies that the second sequence of distributions is used to "correct" the inferences produced using the approximate solver. It is critically important to point out, that the correction required does not depend on the difference $U_f - U_c$ but rather on the variability of the ratio U_f/U_c over the $(\boldsymbol{\theta}, \boldsymbol{d})$ space. Hence if the regions of high-probability of π_c (i.e. the regions where U_c is high) coincide with the high-probability regions under U_f very few (potentially only one) iterations would be needed. The role of the approximate solver is to steer the sampling at regions of interest at a fraction of the cost. Good approximate solvers are therefore those for which the ratio U_f/U_c is as close to a constant as possible over the $(\boldsymbol{\theta}, \boldsymbol{d})$ space. Note that this can be achieved even if $U_f - U_c$ is large. We demonstrate in section 2.3 the reduction in computational cost that can be achieved. Furthermore, if a sequence of increasingly expensive solvers is utilized additional sequences of distributions as in Equation

(2.10) can be defined. If the corresponding densities do not change, the analyst has the option of terminating the sampling. Finally in the case of state augmentation of Equation (2.3), the approximate solver(s) can be readily used at each n by defining two sequences as in Equation (2.9) and Equation (2.10).

2.3 Numerical examples

Both problems examined in this chapter involve random heterogeneous materials in the context of steady-state heat diffusion with a governing PDE:

$$-\nabla \cdot \Big(\lambda(\boldsymbol{x})\nabla T(\boldsymbol{x})\Big) = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega. \tag{2.12}$$

The conductivity field $\lambda(\boldsymbol{x})$ and/or input $f(\boldsymbol{x})$ depend on the vector of uncertainties $\boldsymbol{\theta}$ and design variables $\boldsymbol{d}$. As a result the solution field $T(\boldsymbol{x})$ will also implicitly depend on $(\boldsymbol{\theta}, \boldsymbol{d})$. The finite element method is employed for the discretization of the aforementioned equations leading to the usual system of linear algebraic equations:

$$\boldsymbol{K}(\boldsymbol{\theta}, \boldsymbol{d})\ \boldsymbol{T} = \boldsymbol{F}(\boldsymbol{d}). \tag{2.13}$$

The random variables $\boldsymbol{\theta}$ parameterize the conductivities in a manner to be specialized in each of the examples below along with the boundary conditions.

Before embarking in the presentation of the results, it is worth providing some details on the MCMC kernels used in the Rejuvenation step of the adaptive SMC sampling scheme (Table 1). In particular we employ a Metropolis-within-Gibbs [83] or component-wise Hastings [13] sampler for the $\boldsymbol{\theta}$ and $\boldsymbol{d}$ coordinates separately i.e. we sample from the conditionals $\pi_{\gamma_s}(\boldsymbol{\theta}|\boldsymbol{d}) \propto U^{\gamma_s}(\boldsymbol{\theta}, \boldsymbol{d})p(\boldsymbol{\theta})$ and $\pi_{\gamma_s}(\boldsymbol{d}|\boldsymbol{\theta}) \propto U^{\gamma_s}(\boldsymbol{\theta}, \boldsymbol{d})1_{\mathcal{D}}(\boldsymbol{d})$ respectively (from Equation (2.2)).

Since the latter are analytically unavailable we employ a Metropolis-adjusted Langevin algorithm (MALA, [13]), which for updating the $\boldsymbol{\theta}$-coordinates of a particle i, from $\boldsymbol{\theta}_{s-1}^{(i)}$ to $\boldsymbol{\theta}_{s}^{(i)}$ requires:

$$\boldsymbol{\theta}_{s}^{(i)} = \boldsymbol{\theta}_{s-1}^{(i)} + \frac{\sigma_{\boldsymbol{\theta}}^2}{2}\nabla_{\boldsymbol{\theta}} \log \pi_{\gamma_s}(\boldsymbol{\theta}_{s-1}^{(i)}|\boldsymbol{d}_{s-1}^{(i)}) + \sigma_{\boldsymbol{\theta}}\boldsymbol{Z}_{s-1}^{(i)} \tag{2.14}$$

where $\boldsymbol{Z}_{s-1}^{(i)}$ a vector of i.i.d Gaussian random variables $\boldsymbol{Z}_{s-1}^{(i)} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. The aforementioned proposal was augmented with the standard Metropolis accept/reject step [63]. Two observations are in order. Firstly, application of this scheme requires the calculation of derivatives of the utility function U. Due to its dependence on the solution vector of Equation (2.13) (in a manner to be made specific in the ensuing examples), this entails differentiation of $\boldsymbol{T}(\boldsymbol{\theta}, \boldsymbol{d})$ or the use of adjoint methods. It is noted that such derivatives are also used in deterministic topology optimization schemes [47, 108]. Secondly, the parameter $\sigma_{\boldsymbol{\theta}}$ controls the variance of the random noise. In the simulations performed its value was adjusted at each step of the SMC scheme so as to ensure an acceptance ratio between 50% and 80% [85]. Subsequently, and in order to update the $\boldsymbol{d}$-coordinates of a particle i, from $\boldsymbol{d}_{s-1}^{(i)}$ to $\boldsymbol{d}_{s}^{(i)}$, we employ a MALA scheme:

$$\boldsymbol{d}_{s}^{(i)} = \boldsymbol{d}_{s-1}^{(i)} + \frac{\sigma_{\boldsymbol{d}}^2}{2}\nabla_{\boldsymbol{d}} \log \pi_{\gamma_s}(\boldsymbol{d}_{s-1}^{(i)}|\boldsymbol{\theta}_{s}^{(i)}) + \sigma_{\boldsymbol{d}}\tilde{\boldsymbol{Z}}_{s-1}^{(i)} \tag{2.15}$$

where $\tilde{\boldsymbol{Z}}_{s-1}^{(i)} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. The free parameter $\sigma_{\boldsymbol{d}}$ is also adaptively adjusted so as to ensure an acceptance rate between 50% and 80%.

2.3.1 Designing random heterogeneous materials

This problem is inspired by deterministic topology optimization (as in [108]). The problem domain Ω is the unit square $[0, 1]^2$ and Dirichlet and Neumann boundary conditions are prescribed as in Figure 2.1. The heat source density is assumed constant $f(\boldsymbol{x}) = 1$ as in [108]. Typically in deterministic topology optimization the problem is posed as:

Deterministic topology optimization: *Given two materials with (significantly) different conductivities λ_1 and λ_2 (the values 1 and 0.01 were used in this study) which are to be used with volume fractions V_1 and V_2 (such that $V_1 + V_2 = 1$), find the spatial distribution of the two phases that minimize $J = \int f(\boldsymbol{x})T(\boldsymbol{x})\, d\Omega \approx \boldsymbol{F}^T\boldsymbol{T}$ (the latter objective is referred to as minimum compliance objective in solid mechanics [47]).*

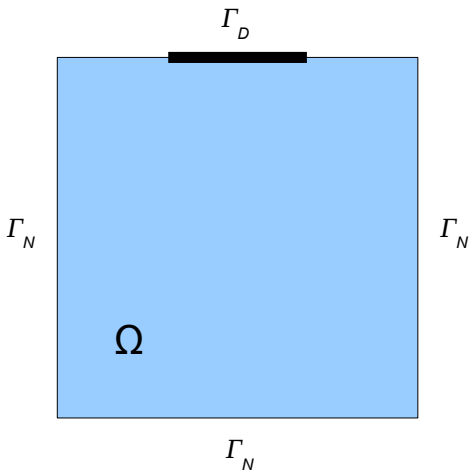


Figure 2.1: Configuration for the problem described in section 2.3.1, taken from [108]. Dirichlet boundary conditions are $T = 0$ on the boundary $\Gamma_D = [0.25, 0.75]$ and Neumann boundary conditions are $-\lambda(\partial T/\partial n) = \mathbf{0}$ on the boundary Γ_N.

In this work, we reformulate the problem for random heterogeneous materials. In particular we consider a composite made up of these two phases with:

$$\lambda(\boldsymbol{x}) = \begin{cases} \lambda_1, & \boldsymbol{x} \in \text{phase 1.} \\ \lambda_2, & \boldsymbol{x} \in \text{phase 2.} \end{cases} \tag{2.16}$$

It is further assumed that $\lambda(\boldsymbol{x})$ is a *random field* whose first-order distribution is fully defined by the volume fractions, i.e. $Pr[\lambda(\boldsymbol{x}) = \lambda_1] = V_1$ and $Pr[\lambda(\boldsymbol{x}) = \lambda_2] = V_2 = 1 - V_1$. Hence the conductivity at each $\boldsymbol{x} \in \Omega$ (or each pixel in the discretized case) is a binary random variable. Clearly, optimizing with respect to $\lambda(\boldsymbol{x})$ and using $\int f(\boldsymbol{x})T(\boldsymbol{x})\, \mathrm{d}\Omega$ as an objective

function is meaningless as they are both random. A viable set of design variables $\boldsymbol{d}$ (given V_1 and V_2) consists of the higher-order statistics of the random field. More concretely, we consider conductivity random fields defined by a mapping from a zero-mean, unit variance Gaussian random field $g(\boldsymbol{x})$ [12, 84] as follows:

$$\lambda(\boldsymbol{x}) = \lambda_1 + (\lambda_2 - \lambda_1)\Phi\left(\frac{g(\boldsymbol{x}) - \mu}{\epsilon}\right) \tag{2.17}$$

where $\Phi(\cdot)$ is the standard normal cumulative distribution function [2]. The threshold μ is selected in order to ensure the desired volume fractions. In this study, for $V_1 = V_2 = 0.5$, the value $\mu = 0$ was used. The higher order statistics of the conductivity field are therefore determined by the covariance function of the Gaussian field $g(\boldsymbol{x})$ [84, 106]. We consider a spectral representation of a statistically homogeneous Gaussian random field with respect to its power spectral density $S_g(\omega_1, \omega_2)$ [91] [3]:

$$\begin{aligned} g\Big(\boldsymbol{x} = (x_1, x_2)\Big) = \sqrt{2}\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1}\Big(&A_{n_1,n_2}\cos(\omega_{1,n_1}x_1 + \omega_{2,n_2}x_2 + \phi_{n_1,n_2}) \\ &+ \tilde{A}_{n1,n_2}\cos(\omega_{1,n_1}x_1 - \omega_{2,n_2}x_2 + \tilde{\phi}_{n_1,n_2})\Big) \end{aligned} \tag{2.18}$$

where $\omega_{1,n_1} = n_1\Delta\omega_1 = n_1\frac{\omega_{1,max}}{N_1}$, $\omega_{2,n_2} = n_2\Delta\omega_2 = n_2\frac{\omega_{2,max}}{N_2}$, [4]

$$A_{n_1,n_2} = \begin{cases} \sqrt{\frac{1}{2}S_g(\omega_{1,n_1}, \omega_{2,n_2})\Delta\omega_1\Delta\omega_2} & \text{if } n_1 = n_2 = 0, \\ \sqrt{S_g(\omega_{1,n_1}, \omega_{2,n_2})\Delta\omega_1\Delta\omega_2} & \text{if } n_1 = 0, n_2 > 0 \text{ or } n_1 > 0, n_2 = 0, \\ \sqrt{2S_g(\omega_{1,n_1}, \omega_{2,n_2})\Delta\omega_1\Delta\omega_2} & \text{otherwise,} \end{cases} \tag{2.19}$$

$$\tilde{A}_{n_1,n_2} = \begin{cases} \sqrt{\frac{1}{2}S_g(\omega_{1,n_1}, -\omega_{2,n_2})\Delta\omega_1\Delta\omega_2} & \text{if } n_1 = n_2 = 0, \\ \sqrt{S_g(\omega_{1,n_1}, -\omega_{2,n_2})\Delta\omega_1\Delta\omega_2} & \text{if } n_1 = 0, n_2 > 0 \text{ or } n_1 > 0, n_2 = 0, \\ \sqrt{2S_g(\omega_{1,n_1}, -\omega_{2,n_2})\Delta\omega_1\Delta\omega_2} & \text{otherwise,} \end{cases} \tag{2.20}$$

and ϕ_{n_1,n_2}, $\tilde{\phi}_{n_1,n_2}$ are random phase angles uniformly distributed in $[0, 2\pi]$.

[2] A very small ϵ is used so that Φ approximates a Heaviside function at μ. In this study $\epsilon = 0.001$.

[3] The power spectral density is the Fourier transform of the autocovariance function.

[4] The upper-cutoff frequencies effectively determine the scale of heterogeneity.

Given the expressions above, the following parameterization is adopted:

- *Random variables*: $\boldsymbol{\theta} = \{\phi_{n_1,n_2}, \tilde{\phi}_{n_1,n_2}\}_{n_1,n_2=0}^{N_1-1,N_2-1}$ of dimension N_1N_2, and uniformly distributed in $[0, 2\pi]$ with $p(\boldsymbol{\theta}) = \left(\frac{1}{2\pi}\right)^{N_1N_2}$.
- *Design variables*: $\boldsymbol{d} = \{S_g(\omega_{1,n_1}, \omega_{2,n_2})\}_{n_1=0,n_2=-(N_2-1)}^{N_1-1,N_2-1}$ of dimension $(2N_2-1)N_1$. It is noted that since we consider unit variance Gaussian fields, the power spectral density must integrate to 1. This imposes a constraint on the sum of the design variables. An additional constraint is that $S_g(\omega_{1,n_1}, \omega_{2,n_2}) \geqslant 0,\ \forall n_1, n_2$.

The objective function $J(\boldsymbol{\theta}, \boldsymbol{d}) = \int f(\boldsymbol{x})T(\boldsymbol{x})\,\mathrm{d}\Omega \approx \boldsymbol{F}^T\boldsymbol{T}$ used in deterministic topology optimization formulations, will now be a random variable due to its dependence on the random spatial distribution of conductivities. The two extreme values it attains are $\sim 10^{-3}$ when $\lambda(\boldsymbol{x}) = \lambda_1 = 1, \forall \boldsymbol{x} \in \Omega$ and $\sim 10^{-1}$ when $\lambda(\boldsymbol{x}) = \lambda_2 = 0.01, \forall \boldsymbol{x} \in \Omega$. The utility function we employ is:

$$U(\boldsymbol{\theta}, \boldsymbol{d}) = \begin{cases} 1 & if\ J(\boldsymbol{\theta}, \boldsymbol{d}) \leqslant J_0 = 2.5 \times 10^{-3}, \\ e^{-c(J(\boldsymbol{\theta},\boldsymbol{d})-J_0)} & if\ J(\boldsymbol{\theta}, \boldsymbol{d}) > J_0 = 2.5 \times 10^{-3}. \end{cases} \tag{2.21}$$

For a large c (the value $c = 10^4$ was used in this study) the aforementioned utility function approximates the Heaviside function at $J(\boldsymbol{\theta}, \boldsymbol{d}) = J_0$. As a result the expected utility $\hat{U}(\boldsymbol{d})$ (Equation (2.1)) represents the *probability that J is less than the threshold J_0*. Hence, the formulation of the problem in the stochastic topology optimization framework proposed is:

Stochastic topology optimization: *Given two materials with (significantly) different conductivities λ_1 and λ_2 (the values 1 and 0.01 were used in this study) which are to be* randomly *distributed with volume fractions $V_1 = V_2 = 0.5$, find the* random spatial distribution *as prescribed by the design variables $\boldsymbol{d}$ above, that maximize the probability that $J(\boldsymbol{\theta}, \boldsymbol{d})$ is less than a prescribed threshold J_0.*

In order to provide insight to the goals and results of the stochastic topology optimization proposed, we first consider a simplified power spectral density given by:

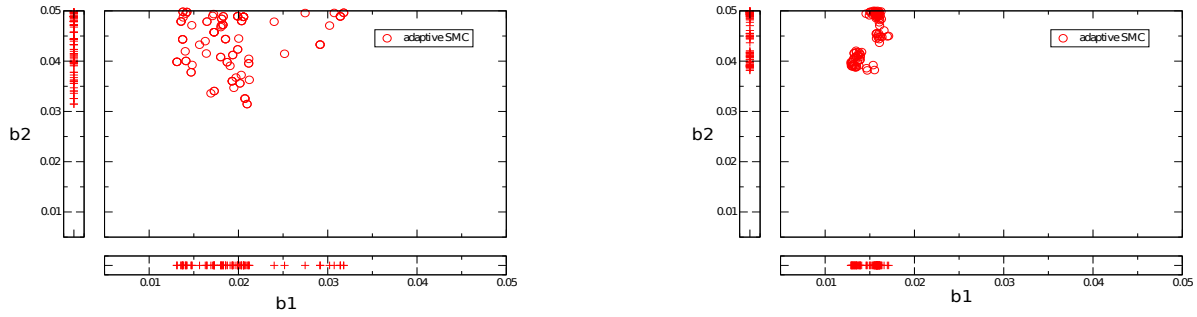
$$S_g(\omega_1, \omega_2) = \frac{b_1 b_2}{4\pi} \exp\left(-\left(\frac{b_1\omega_1}{2}\right)^2 - \left(\frac{b_2\omega_2}{2}\right)^2 \right). \tag{2.22}$$

The corresponding autocovariance $C_g(\Delta x_1, \Delta x_2)$ and Fourier transform pair with S_g is given by:

$$C_g(\Delta x_1, \Delta x_2) = \exp\left(-\left(\frac{\omega_1}{b_1}\right)^2 - \left(\frac{\omega_2}{b_2}\right)^2 \right) \tag{2.23}$$

where the role of b_1 and b_2 can be clearly seen as the *correlation lengths* in the two dimensions. We are assuming the form of Equation (2.22) for the spectral density reduces the design variables to two, i.e. $\boldsymbol{d} = (b_1, b_2)$. In the simulations performed, $N = 100$ particles were used and the range of values of the design variables examined was $\mathcal{D} = [0.005, 0.05] \times [0.005, 0.05]$. We employed $\Delta\omega_1 = \Delta\omega_2 = 2\pi$ and $N_1 = N_2 = 64$ in Equation (2.18). Figure 2.2 depicts the particle locations with respect to the design variables with $n = 1$ and $n = 2$ state augmentations. The former implies sampling in $8,192 + 2 = 8,194$ dimensions and the latter in $16,384 + 2 = 16,386$ dimensions (where $8,192$ is the dimension of the uncertainties $\boldsymbol{\theta}$ in Equation (2.18)). As it can be clearly seen the maximum utility is attained for $b_1^* \approx 0.015 < b_2^* \approx 0.045$.

The result of the *deterministic topology optimization* is shown in Figure 2.3(a) [108]. The high-conductivity material is depicted with black ($\lambda_1 = 1$) and mostly occupies the region under the boundary Γ_D where Dirichlet boundary conditions are specified. Thick or thin clusters of the phase 1 emanate from Γ_D over the whole domain in order for the energy introduced by the heat source $f(\boldsymbol{x})$ (Equation (2.12)) to permeate throughout Ω, ensuring a low value of the objective. Obviously such realizations (as in Figure 2.3(a)) are inconsistent with a statistically homogeneous random field $\lambda(\boldsymbol{x})$ prescribed in the stochastic optimization framework. For example it is clear that the volume fraction of the black phase is close to 1



(a) $n = 1$: Sampling in $8,192 + 2$ dimensions

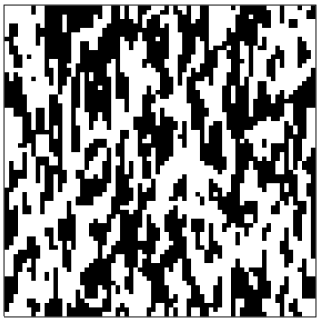
(b) $n = 2$: Sampling in $16,384 + 2$ dimensions

Figure 2.2: Particles $\{\boldsymbol{d}^{(i)} = (b_1^{(i)}, b_2^{(i)})\}_{i=1}^{N=100}$ for the stochastic topology optimization problem using the spectral density of Equation (2.22)

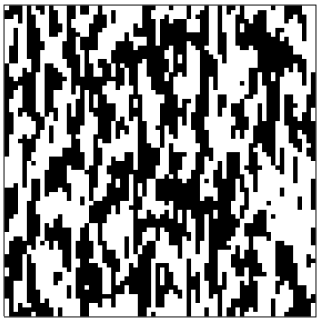
near Γ_D and drops to zero far away. The proposed framework however allows one to control microstructures by controlling the statistics of their random distribution (in this case through S_g). Hence the composite is always random and its first-order distribution (as expressed by the volume fractions) is always the same. Figures 2.3(b)-2.3(c)-2.3(d) depict three realizations of the random medium generated by the optimal b_1^*, b_2^* identified in Figure 2.2(b). It can be clearly seen that the *optimal random microstructures* are statistically anisotropic. As it is perhaps expected from the result of the deterministic topology optimization (Figure 2.3(a)) and since $b_2 > b_1$ (Figure 2.2), the optimal random composite exhibits higher connectivity in the vertical direction x_2 than in the horizontal x_1.

We also considered the general case, where the design variables consist of the values of the S_g at the frequencies in Equation (2.18), i.e. $\boldsymbol{d} = \{S_g(\omega_{1,n_1}, \omega_{2,n_2})\}_{n_1=0,n_2=-(N_2-1)}^{N_1-1,N_2-1}$ The simulations were carried out with $N = 100$ particles and $n = 5$ state augmentations (Equation (2.3)). Since the design variables $\boldsymbol{d}$ represent the power spectral density S_g, the particulate approximations (i.e. particle values $\boldsymbol{d}^{(i)}$ and weights $W^{(i)}$) were used to estimate the expected value of S_g as in Figure 2.4(a). The corresponding autocovariance function C_g

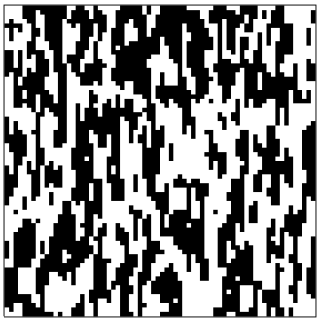
(a) Results from deterministic topology optimization



(b) Realization 1



(c) Realization 2

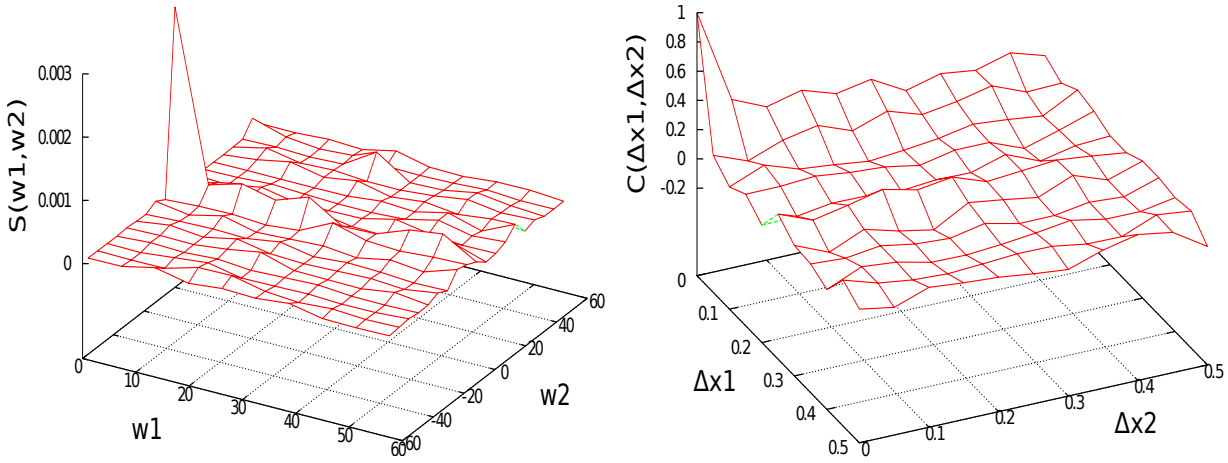


(d) Realization 3

Figure 2.3: (a) Result of *deterministic* topology optimization, taken from [108] (Copyright ©2009 Society for Industrial and Applied Mathematics.) (b), (c), (d) Sample realizations obtained from the proposed stochastic topology optimization scheme. All results correspond to a 70×70 grid. The high-conductivity material is depicted in black ($\lambda_1 = 1$).

is depicted in Figure 2.4(b). It is noted that the latter *is different* from the autocovariance C_λ of the resulting binary field $\lambda(x)$ in Equation (2.17). It can be established [60] that for this particular example with volume fraction 50% (and for $\epsilon \to 0$ in Equation (2.17)):

$$C_\lambda(\Delta x_1, \Delta x_2) = \frac{(\lambda_2 - \lambda_1)^2}{2\pi} \arcsin C_g(\Delta x_1, \Delta x_2). \tag{2.24}$$



(a) Spectral density $S_g(\omega_1, \omega_2)$

(b) Autocovariance

Figure 2.4: (a) Average power spectral density $S_g(\omega_1, \omega_2)$ found by averaging over the particles i.e. $\sum_{i=1}^{N} W^{(i)} \boldsymbol{d}^{(i)}$ (Equation (2.18) for $N_1 = N_2 = 10$ and $\omega_{1,max} = \omega_{2,max} = 20\pi$) (b) Corresponding autocovariance $C_g(\Delta x_1, \Delta x_2)$ for various separations $(\Delta x_1, \Delta x_2)$, found by taking the Fourier transform of $S_g(\omega_1, \omega_2)$ in (a)

Figures 2.5(a)-2.5(b)-2.5(c) depict three realizations of the random medium generated by the optimal S_g shown in Figure 2.4(a) through the mapping of Equation (2.17). It is noted that the average S_g corresponding to a uniform distribution on the design variables gives rise to random checkerboards, i.e. realizations that exhibit zero correlation. The realizations of Figures 2.5(a)-2.5(b)-2.5(c) however exhibit strong correlation patterns. In particular it is noted that the black phase (high conductivity) exhibits connected paths, particularly in the vertical direction (x_2). This is also verified in Figure 2.6 where the lineal-path functions of the black phase for various separations in the directions x_1 (horizontal) and x_2 (vertical) have

been calculated. In the pixelized version, the lineal-path function $L(\Delta x)$ [106] expresses the probability that a line of pixel-length Δx lies wholly on the black phase. It can be readily calculated using the spectral density S_g (Figure 2.4(a), or the autocovariance (Figure 2.4(b)) of the underlying Gaussian field in Equation (2.17). As in the previous simplified scenario (Figure 2.3), the *optimal random microstructure* is statistically anisotropic and exhibits higher connectivity in the vertical direction x_2 than in the horizontal x_1.

(a) Realization 1 (b) Realization 2 (c) Realization 3

Figure 2.5: Sample realizations obtained from the spectral density of Figure 2.4(a) found by the proposed stochastic topology optimization scheme. All results correspond to a 70×70 grid. The high-conductivity material is depicted in black ($\lambda_1 = 1$).

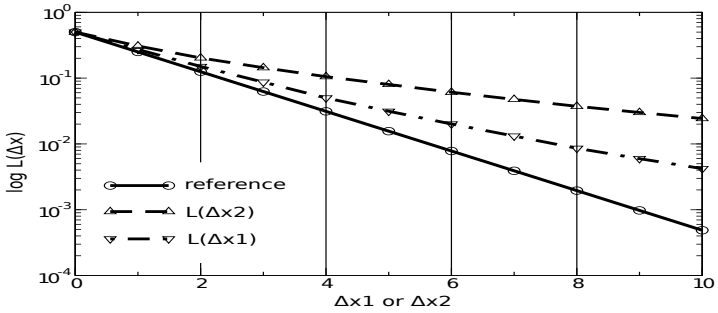


Figure 2.6: Lineal path functions in the directions x_1 and x_2 for various separations Δx_1 or Δx_2. These are compared with the linear path function of a random checkerboard, i.e. with no correlation.

2.3.2 Design/Control of random heterogeneous materials

The goal of this problem is to optimally select the input in a random system described by a heterogeneous medium so as to maximize an expected utility related to the response. In particular we consider the rectangular domain $\Omega = [-1, 1] \times [0, 1]$ of Figure 2.7, where $T = 0$ on the right boundary Γ_D, $-\lambda(\boldsymbol{x})(\partial T(\boldsymbol{x})/\partial n) = \mathbf{0}$ on the top and bottom boundaries Γ_{N_1} and $-\lambda(\boldsymbol{x})(\partial T(\boldsymbol{x})/\partial n) = \boldsymbol{q}$ on the left boundary Γ_{N_2}.

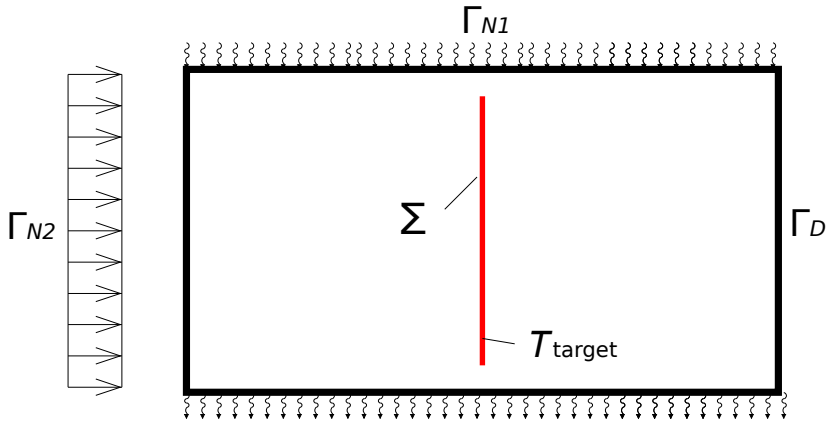


Figure 2.7: Configuration of the problem described in section 2.3.2

The design variables $\boldsymbol{d}$ in this problem parameterize the imposed flux $\boldsymbol{q}$ on the left boundary Γ_{N_2}. We consider several utility functions that relate to the temperature profile $\boldsymbol{T}_\Sigma$ along the vertical line Σ located at $x_1 = 0$ [112]. The random variables $\boldsymbol{\theta}$ parameterize the random conductivity field $\lambda(\boldsymbol{x})$. In particular we consider a statistically homogeneous $\lambda(\boldsymbol{x})$ defined through a zero-mean Gaussian field $g(\boldsymbol{x})$ as:

$$\lambda(\boldsymbol{x}; \boldsymbol{\theta}) = e^{g(\boldsymbol{x})}. \tag{2.25}$$

The autocovariance $C(\Delta x_1, \Delta x_2)$ of $g(\boldsymbol{x})$ is prescribed as:

$$C(\Delta x_1, \Delta x_2) = \exp\left(-\frac{\Delta x_1^2 + \Delta x_2^2}{x_0^2}\right). \tag{2.26}$$

The value of $x_0 = 0.2$ for the correlation length was used in this study. In order to obtain a resolution-independent representation of $g(\boldsymbol{x})$ we employed a Karhunen-Loève expansion:

$$g(\boldsymbol{x}) = \sum_{i=1}^{n_\theta} \theta_i \sqrt{\mu_i}\phi_i(\boldsymbol{x}) \tag{2.27}$$

where the eigenvalues μ_i and eigenfunctions $\phi_i(\boldsymbol{x})$ of the autocovariance can be calculated semi-analytically as in [42]. The series was truncated at $n_{\boldsymbol{\theta}} = 1,000$ which was found to represent 95% of the variance of $g(\boldsymbol{x})$. The corresponding $1,000$ standard normal variates θ_i represent the random variables $\boldsymbol{\theta}$ in this problem. Figure 2.8 depicts a sample realization of $\lambda(\boldsymbol{x})$ obtained from Equations (2.25) and (2.27).

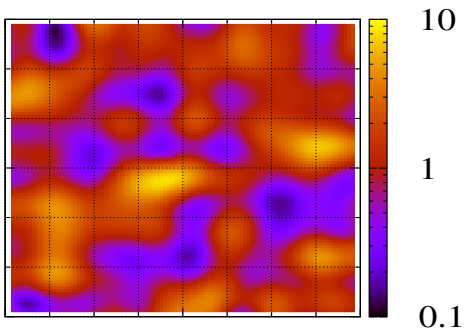


Figure 2.8: Sample realization of the conductivity field $\lambda(\boldsymbol{x})$ prescribed in Equations (2.25) and (2.27)

One design variable – Bimodal expected utility

In this example we consider a constant flux on the left boundary Γ_{N_2} and therefore a single design variable. The utility function employed was:

$$U(\boldsymbol{\theta}, \boldsymbol{d}) = \exp\left(-\frac{\| \boldsymbol{T}_\Sigma - \boldsymbol{T}^{(1)}_{target} \|^2}{2\sigma^2}\right) + 6\exp\left(-\frac{\| \boldsymbol{T}_\Sigma - \boldsymbol{T}^{(2)}_{target} \|^2}{2\sigma^2}\right). \tag{2.28}$$

Each of the terms in the sum above provide a measure of the difference between the temperature profile $\boldsymbol{T}_\Sigma$ along Σ (see Figure 2.7) and some target values $\boldsymbol{T}^{(1)}_{target}$ and $\boldsymbol{T}^{(2)}_{target}$.

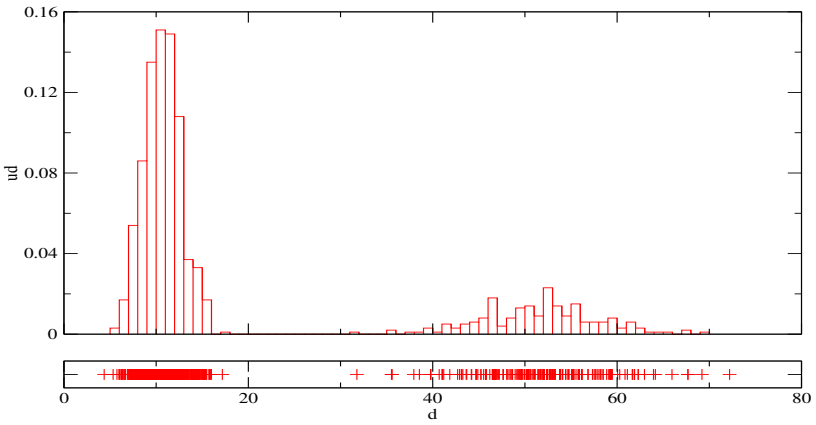
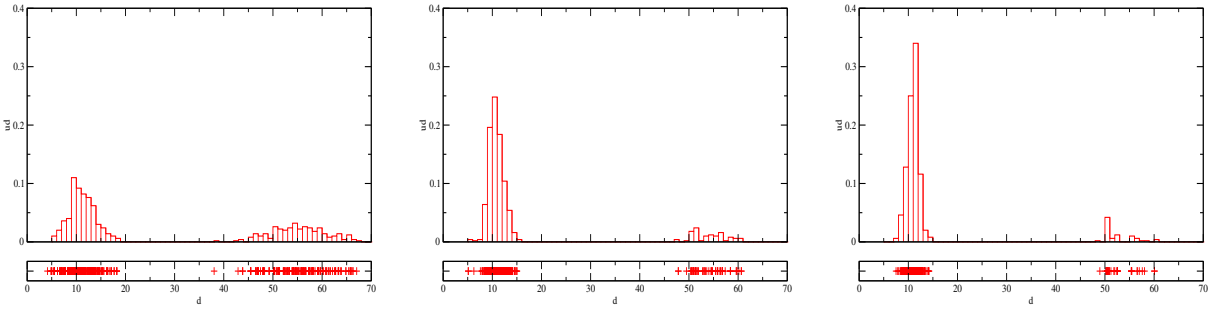


Figure 2.9: Particles $\{\boldsymbol{d}^{(i)}\}_{i=1}^{N=1,000}$ and corresponding histogram. The latter is proportional to the expected utility $\hat{U}(\boldsymbol{d})$ of the problem described in section 2.3.2.

Hence we seek the imposed flux so that the temperatures $\boldsymbol{T}_\Sigma$ are as close as possible to the prescribed targets. It is noted that $\boldsymbol{T}_\Sigma$ depends on $\boldsymbol{\theta}$ and $\boldsymbol{d}$ but this has been omitted for notational economy. The reason two target profiles were selected is to assess whether the proposed scheme can correctly identify more than one local maxima of the expected utility. It is finally mentioned that the value $\sigma = 5$ was used and the target temperature profiles were constant along Σ and equal to 35 and 75 respectively.

Figure 2.9 depicts the $\boldsymbol{d}$ coordinates of the particles and their histogram which is proportional to the expected utility $\hat{U}(\boldsymbol{d})$ (Equation (2.1)). This simulation was performed with $N = 1,000$ particles and entailed sampling in $n_{\boldsymbol{\theta}} + n_{\boldsymbol{d}} = 1,000 + 1 = 1,001$ dimensions. The algorithm can clearly identify and populate the two modes which correspond to two distinct local maxima of the expected utility, despite the high-dimensionality of the sampling space.

The same utility function was used and state augmentation was employed in order to test the capability of the algorithm to zoom in the global maximum. Figure 3.8 depicts particles and expected utilities with $n = 1$, $n = 3$ and $n = 5$ state augmentations as in Equation (2.3). It is clearly seen that the global maximum is identified.



(a) $n = 1$: Sampling in $1,001$ dimensions

(b) $n = 3$: Sampling in $3,001$ dimensions

(c) $n = 5$: Sampling in $5,001$ dimensions

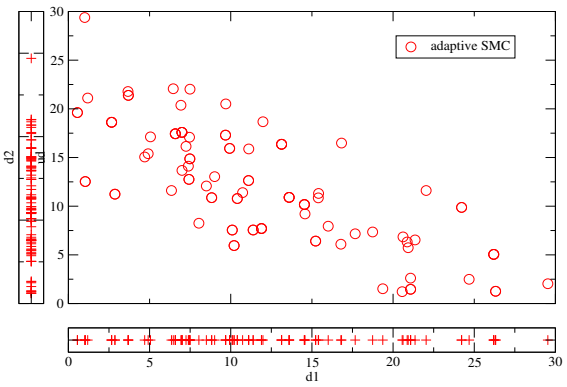
Figure 2.10: Particles $\{\boldsymbol{d}^{(i)}\}_{i=1}^{N=1,000}$ and corresponding histograms for numbers $n = 1, 3, 5$ state augmentations for the utility function of Equation (2.28)

Two design variables – Unimodal expected utility

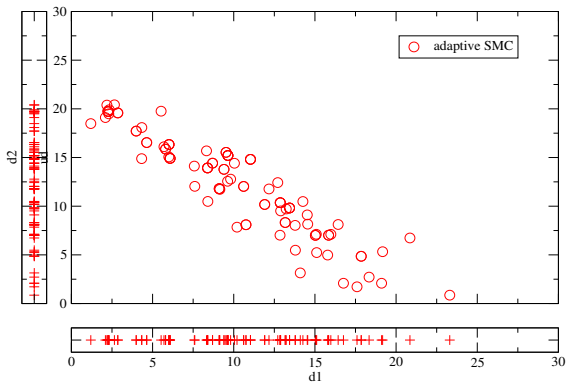
In this problem we increase the design variables to two i.e. $\boldsymbol{d} = (d_1, d_2)$. In particular it is assumed that d_1 represents the flux on the upper half of Γ_{N_2} and d_2 at the lower. The following utility function was used:

$$U(\boldsymbol{\theta}, \boldsymbol{d}) = \exp\left(-\frac{\| \boldsymbol{T}_\Sigma - \boldsymbol{T}_{target} \|^2}{2\sigma^2}\right) \tag{2.29}$$

and the target temperature profile was taken constant and equal to 35. Figure 2.11 depicts the $\boldsymbol{d}$-coordinates of $N = 100$ particles that were used to solve this problem without ($n = 1$) and with state augmentation ($n = 5$). These runs entailed sampling in $1,002$ and $5,002$ dimensions respectively. An interesting observation arising from these results is that the maxima of the expected utility seem to be attained for $d_1 + d_2 = 20$. This is more clear in Figure 2.11(b) and provides physical insight into the sensitivity of the random system to the input $\boldsymbol{d}$. In particular, it appears that the total flux (i.e. $d_1 + d_2$) controls the temperature profile along Σ.



(a) $n = 1$: Sampling in $1,002$ dimensions



(b) $n = 5$: Sampling in $5,002$ dimensions

Figure 2.11: Particles $\{\boldsymbol{d}^{(i)}\}_{i=1}^{N=100}$ for the utility function of Equation (2.29)

Multi-resolution analysis

The goal of this last example is to illustrate the potential of significant computational savings by employing approximate models in the manner explained in section 2.2.2. In particular, we consider a two-resolution approach where the role of the approximate model is played by a finite element solver with a coarse resolution. Our coarse/approximate model consists of 200 triangular finite elements and our reference/fine solver of 800 finite elements (in both cases uniform meshes were used). The comparisons in terms of computational cost are expressed in terms of the (equivalent) number of calls to the finer (most expensive) solver. In this problem, the cost of the coarse solver is much less and corresponds to 1/64 calls to the fine solver. Our goal is to use the former in order to expedite the sampling and reduce the overall number of calls to the latter solver.

We consider a single design variable $\boldsymbol{d}$ (representing the flux on Γ_{N_2}) and the utility function of Equation (2.29). The cost of the reference solution which employs only the fine solver and the advocated adaptive SMC scheme is $7,200$ calls. The result of this simulation in terms of the $\boldsymbol{d}$-coordinates of the $N = 100$ particles used and the expected utility $\hat{U}(\boldsymbol{d})$

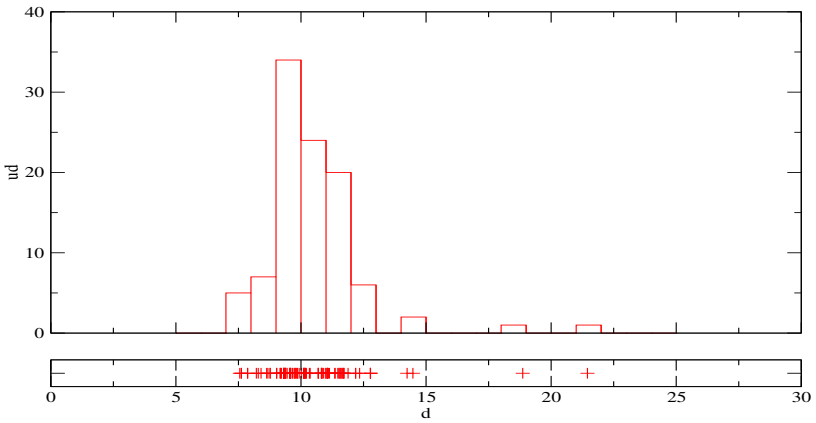


Figure 2.12: Particles $\{\boldsymbol{d}^{(i)}\}_{i=1}^{N=100}$ and corresponding histogram. The latter is proportional to the expected utility $\hat{U}(\boldsymbol{d})$ in section 2.3.2.

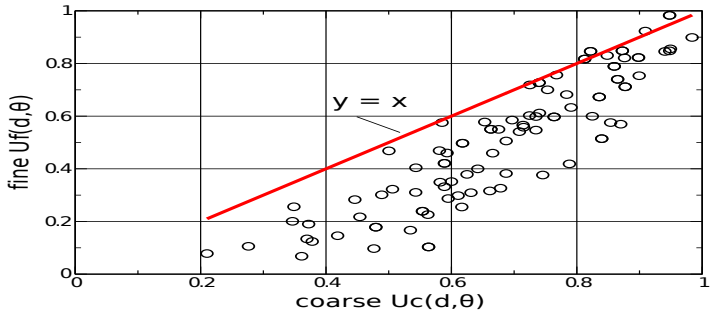


Figure 2.13: Each circle corresponds to $(U_c(\boldsymbol{\theta}, \boldsymbol{d}), U_f(\boldsymbol{\theta}, \boldsymbol{d}))$ (given by Equation (2.29)) evaluated using the coarse and fine solvers for the same $(\boldsymbol{\theta}, \boldsymbol{d})$ values each time.

are depicted in Figure 2.12.

Figure 2.13 compares the value of the utility function (Equation (2.29)) calculated for the same $\boldsymbol{\theta}$ and $\boldsymbol{d}$ values using the coarse i.e. U_c and fine, i.e. U_f solvers (section 2.2.2). It can be seen that the coarse model underestimates U_f and in absolute terms provides a poor approximation. Furthermore one observes a significant scatter which clearly implies that U_c cannot uniquely predict U_f.

In the proposed framework however, as explained in section 2.2.2, it suffices that the output U_c of the coarse/approximate model is correlated with U_f in order to achieve computational savings. We employed the two sequences of distributions as in Equation (2.9) and Equation (2.10). The first allows us to estimate the maxima of the coarse/approximate expected utility and requires only calls to the inexpensive/coarse solver. The $\boldsymbol{d}$-coordinates and the estimated expected approximate utility are depicted in Figure 2.14 (blue line – "coarse"). The cost of obtaining this result with $N = 100$ particles was equivalent to 138 calls to the fine/expensive solver. Obviously the result differs from the expected fine/reference utility (red line – "fine" in Figure 2.14). Using this distribution as the starting point for sampling from the second sequence of distributions in Equation (2.10), which requires calls to the coarse and fine solvers, ultimately leads to the result depicted in Figure 2.14) (green line – "coarse+fine"). The cost of sampling from this second sequence was equivalent to 717 runs of the most expensive solver. Hence even though the total cost was $138 + 717 = 955$ runs, i.e. a reduction by a factor of 7, the result obtained practically coincides with the reference solution.

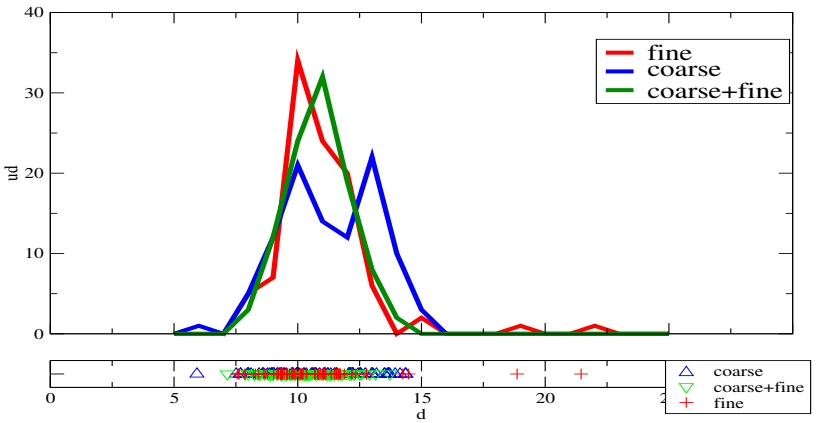


Figure 2.14: Particles $\{\boldsymbol{d}^{(i)}\}_{i=1}^{N=100}$ and corresponding histograms obtained using only the fine solver (red), only the coarse solver (blue) ,and a combination of coarse and fine solvers (green) as in the sequences of Equation (2.9) and Equation (2.10).

The efficiency gain by utilizing the approximate model was also tested for the utility function of Equation (2.28). As it can be seen in Figure 2.15, which compares U_c and U_f, the coarse solver provides a very poor approximation of the output of the fine solver in terms of U_f. It is also noted that the quality of the approximation seems to deteriorate for large utility function values which are of interest.

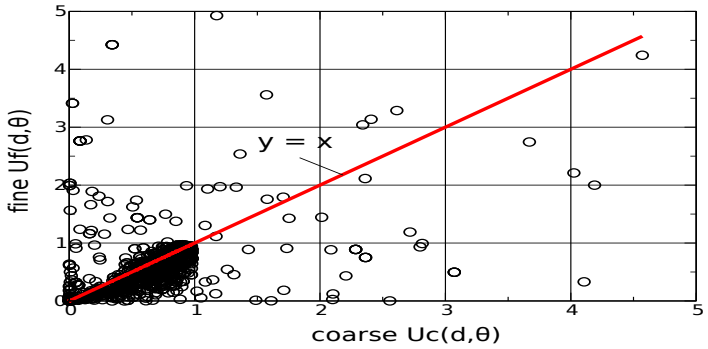


Figure 2.15: Each circle corresponds to $(U_c(\boldsymbol{\theta}, \boldsymbol{d}), U_f(\boldsymbol{\theta}, \boldsymbol{d}))$ (given by Equation (2.28)) evaluated using the coarse and fine solvers for the same $(\boldsymbol{\theta}, \boldsymbol{d})$ values each time.

Figure 2.16 compares the accuracy of the sampling with $N = 1,000$ particles. It is noted that the cost of using exclusively the fine solver (red line – "fine") is equivalent to $57,000$ runs whereas the total cost of employing the coarse and fine solver in the manner explained in section 2.2.2 is $25,500$ runs. Even though the computational savings achieved (a factor of 2) are not as striking as in the previous case, it is still significant, particularly when considering the poor correlation between the two outputs in Figure 2.15. The expected utilities estimated exhibit negligible differences (red and green lines in Figure 2.16) despite the presence of two modes.

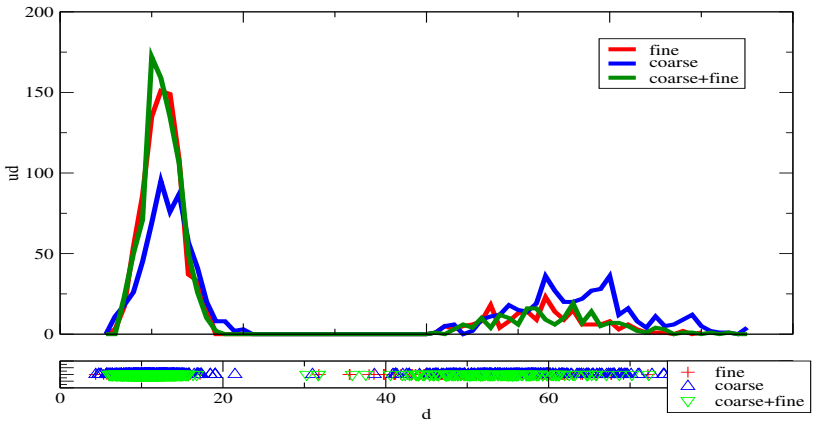


Figure 2.16: Particles $\{\boldsymbol{d}^{(i)}\}_{i=1}^{N=1,000}$ and corresponding histograms obtained using only the fine solver (red), only the coarse solver (blue), and a combination of coarse and fine solvers (green) as in the sequences of Equation (2.9) and Equation (2.10).

2.4 Conclusions

A critical task in the context of random heterogeneous materials involves their design and the optimization of their response/behavior/performance. An embarrassingly parallelizable sampling scheme is discussed that is capable of dealing with systems with high-dimensional vectors of uncertainties and design variables. An efficient adaptive SMC scheme is proposed that can efficiently populate regions of the design space where the expected utility function attains its maxima. The proposed framework allows the principled utilization of approximate models in order to achieve further reduction in computational cost and enables the multi-resolution analysis of such problems. A possible extension involves the utilization of statistical regression techniques to identify the relation between the approximate and the reference models that could lead to more efficiency gains.

CHAPTER 3

REDUCED ORDER MODEL TRACKING AND INTERPOLATION TO SOLVE PDE-BASED BAYESIAN INVERSE PROBLEMS

Abstract

This chapter presents a computationally efficient probabilistic framework that enables the identification of model parameters from noisy measurements of the response. We consider transient PDE-based models, where the parameters correspond to physical properties. An efficient and reliable procedure for estimation of those unknown parameters is pursued. The proposed framework uses a Bayesian approach, an efficient Sequential Monte Carlo sampling scheme, and adaptive reduced order models. The Bayesian approach has several advantages including the ability to provide not only point estimates of the quantities of interest, but also measures of credibility and robustness concerning those estimates. The associated Sequential Monte Carlo method sampling scheme is embarrassingly parallelizable, as well as efficient in terms of the number of calls to the forward solver (*e.g.* a finite element code) used in evaluating the likelihood function. We propose to use a reduced order model (ROM) adaptation procedure where projection-based ROMs are seen as points on a certain Riemannian manifold and are "tracked" and interpolated during the sampling process using a database of precomputed ROMs. This approach ensures that an appropriate ROM is used for the likelihood evaluation in every region of the parameter space, thus leading to both significant computational savings and improvements in accuracy. Using numerical examples, we illustrate the capabilities of the proposed framework, and show that it leads to quality estimates with a quantified predictive uncertainty.

3.1 Introduction

The dimension of Computational Science and Engineering (CSE) involving the rigorous and principled quantification of uncertainty (concerning predictive abilities of computer models or simulations, *vis-à-vis* their real world counterparts) is an area of investigation receiving considerable attention within the modern scientific literature [73]. When approaching the problem of uncertainty quantification (UQ) in CSE, it is inevitable that computational demands grow; as compared with more traditional approaches involving deterministic strategies for simulation. One important class of approaches supporting UQ in CSE involves Monte Carlo strategies [63]. It is one version within the Monte Carlo approaches that is considered in the present work, and more thoroughly described later herein. The inherent computational demands that accompany Monte Carlo strategies motivates the development of a reduced order modeling (ROM) framework that is expedient within contexts involving a Bayesian description of the underlying uncertain system considered. Application of this ROM framework within a Bayesian context, to enable UQ with Monte Carlo simulation, is the major contribution of the present work.

The proposed ROM framework adopts a strategy wherein the numerical description of a computational analog to a real world system is *projected* onto a suitably selected subspace, such that the resulting system is much less computationally demanding to solve than the original, unprojected system. Such projection schemes are not new, and the interested reader is directed to [74], and references therein, for a helpful survey of the salient literature. While such projection schemes for ROM construction can yield very good results in practically useful engineering situations [75, 15], robustness can sometimes be an issue when details of subsequent model definitions deviate from the underlying assumptions adopted during the initial ROM construction. This is an important shortcoming to consider when seeking

to apply projection-based ROM strategies to ameliorate computational demands accompanying the application of Mote Carlo strategies in support of UQ. Inevitably, Monte Carlo approaches will require the consideration of a simulation instance that falls outside of the the parameters considered within the initial ROM construction context; thus calling into question the accuracy of any resulting models. However, It is possible to consider strategies for "interpolating" among a database of selected projection operators, as a means for easing such concerns.

Again, such "interpolation" strategies for projection-based ROM construction are not new, and a recent contribution to the literature [26] furnishes an excellent summary of recent activity on this front. However, there are two important works that are not included in that survey of the literature, and these are described now. Amsallem et al. extend their earlier approach for "interpolating" ROM bases (by following geodesics on the Grassmann manifold, whose elements are represented within the orthogonal Stiefel manifold, via interpolation within the tangent space [6]) for use in a special, but perennially useful case in structural dynamics where the system is not forced [5]. Within this particular context, it then becomes possible to "interpolate", directly, the linear ROM operators themselves (not simply interpolating the ROM bases.) In such a case, the Riemannian manifold of interest is merely the space of all symmetric positive definite matrices, and so, in many ways, the details concerning the proposed "ROM interpolation" are simpler to implement than the "ROM basis interpolation" proposed earlier by these same investigators [6]. Of course, this direct ROM interpolation is only possible in restricted contexts, and so the present work adopts the more robust, earlier approach of Amsallem et al. [6].

Another very important work to mention is the recent contribution [16]: wherein a *reduced basis* approach [3, 72, 71, 79, 80, 66] is adapted for variational problems whose defining coefficients are stochastic. In this recent work, the investigators develop rigorous *a poste-*

riori estimates for the statistical outputs of the uncertain variational problem, and thus enable a robust and reliable approach for quantifying uncertainty regarding a specific class of problems: those whose parameterization of the weak form is expressible as the sum of parameter dependent functions and parameter independent forms [16]. For other classes of problems (*e.g.* where parameter dependence is nonlinear), this approach will not be robust, thus motivating the currently proposed approach to ROM enabled UQ.

The present chapter is concerned with the identification of salient model parameters (serving to describe an engineering system of interest) by solving a stochastic inverse problem whose *ground truth* condition emanates from sparse, noisy system response measurements. A Bayesian context is adopted, and the resulting *posterior distribution* of the unknown model parameters is sampled from, dependently, using Monte Carlo strategies, to be described in the sequel. Examples using this stochastic, Bayesian approach can be found in [109, 67, 64, 62]. It is within this setting that reduced order model tracking and interpolation strategies will be employed to ease the computational demands associated with repeated calls to high fidelity computational simulations required as part of an evaluation of the *likelihood*; as needed within the Bayesian framework of our inverse problems. The novelty of the present work rests in the demonstration of the potential for accelerating inverse solution times when quantifying uncertainty in model parameter identification problems. By employing ROM tracking and interpolation, in conjunction with *Sequential Monte Carlo Sampling*, considerable reduction in solution times are enjoyed.

We begin by describing the current problem formulation in Section 3.2: both in terms of the uncertain system considered, and the form of the underlying stochastic inverse problem. Section 3.3 goes on to describe the important details concerning adaptive sequential Monte Carlo approaches adopted in the work, in addition to furnishing a description of the ROM tracking and interpolation approach employed herein. Section 3.4 develops two example

problems used to demonstrate the current approach, and subsequent conclusions are drawn in Section 3.5.

3.2 Problem formulation

We consider a time-dependent system described by a system of partial differential equations defined on $\mathbb{R}^d \times \mathbb{R}^+$, $d = 1, 2, 3$ being the number of spatial dimensions:

$$\mathcal{L}(\boldsymbol{u}(\boldsymbol{x}, t; \boldsymbol{\theta}); \boldsymbol{\theta}) = 0.$$

The differential operator $\mathcal{L}$ and the system response $\boldsymbol{u}(\boldsymbol{x}, t; \boldsymbol{\theta})$ are both dependent on a set of P model parameters of interest $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_P)$. We aim to determine these parameter values from sparse, noisy measurements of the system response $\boldsymbol{u}_{i,j}^{\text{obs}}$ at various locations $\boldsymbol{x}_i$ and times t_j.

We will assume a linearly additive noise model for the measurements, of the form

$$\boldsymbol{u}_{i,j}^{\text{obs}} = \boldsymbol{u}_{i,j}^{\text{FE}}(\boldsymbol{\theta}) + \boldsymbol{\eta}_{i,j} \tag{3.1}$$

where $\boldsymbol{u}_{i,j}^{\text{FE}}(\boldsymbol{\theta})$ is some assumed suitable deterministically modeled response at location i and time j, as instantiated by the collection of model parameters $\boldsymbol{\theta}$. It will be assumed that the noise values at the M spatio-temporal points i, j are independent and identically distributed (i.i.d.) from a zero-mean normal distribution of variance σ^2, implying the following form of the marginal likelihood:

$$f\Big(\boldsymbol{u}_{i,j}^{\text{obs}} | \boldsymbol{\theta}\Big) = \frac{1}{\sigma\sqrt{2\pi}} \exp\bigg(-\frac{1}{2\sigma^2} \Big\| \boldsymbol{u}_{i,j}^{\text{obs}} - \boldsymbol{u}_{i,j}^{\text{FE}}(\boldsymbol{\theta}) \Big\|^2 \bigg).$$

Denoting by $\boldsymbol{y}^{\text{obs}} = \Big\{ \boldsymbol{u}_{i,j}^{\text{obs}} \Big\}$ the set of M observed measurements and exploiting the i.i.d. assumption, the global likelihood is given by

$$f\Big(\boldsymbol{y}^{\text{obs}} | \boldsymbol{\theta}\Big) = \frac{1}{(\sigma\sqrt{2\pi})^M} \exp\bigg(-\frac{1}{2\sigma^2} \sum_{i,j} \Big\| \boldsymbol{u}_{i,j}^{\text{obs}} - \boldsymbol{u}_{i,j}^{\text{FE}}(\boldsymbol{\theta}) \Big\|^2 \bigg). \tag{3.2}$$

The likelihood function describes the probability of having observed a specific sequence of values $\boldsymbol{y}^{\text{obs}}$ for a given set of parameters $\boldsymbol{\theta}$.

Maximum Likelihood Estimation (MLE) can be performed to find the parameters that maximize the probability of having observed the measured values. Gradient-based optimization and regularization techniques are mainly used towards this goal, generally by maximizing the logarithm of the likelihood; thus resulting in a single estimated value of a local maximum. A Bayesian approach is adopted herein, which leads not only to a point estimate of the parameters of interest but also a probability distribution [82, 39]; thus enabling one to identify the different maxima, while at the same time providing valuable measures of sensitivity and robustness concerning those values. We stress here the important change of paradigm, as a component from the parameter vector $\boldsymbol{\theta}$ is now seen as a random variable, of which its conditional distribution, given the data, is to be inferred. By attributing a prior distribution to the model parameters and applying Bayes' Theorem, we obtain their posterior distribution; which enables one to quantify the uncertainty concerning the parameter values.

The application of the aforementioned Bayes' Theorem enables one to uncover the desired posterior distribution of the parameters given the acquired data, thus providing the "solution" to the inverse problem

$$\pi\left(\boldsymbol{\theta}|\boldsymbol{y}^{\text{obs}}\right) \propto f\left(\boldsymbol{y}^{\text{obs}}|\boldsymbol{\theta}\right)\pi(\boldsymbol{\theta}) \tag{3.3}$$

where $\pi(\boldsymbol{\theta})$ is the so-called prior distribution, encapsulating any available prior information concerning the parameters. If no information is readily available beforehand, a uniform or a more robust non-informative prior can still be used [55]. In problems in which data are collected sequentially, the posterior with the previous data can be readily used as the prior for the subsequent data points. Note that by taking the logarithm of the previous relation, the prior can also be interpreted as a classical Tikhonov regularization term [54]. Specific

prior choices will be discussed in section 3.4.

Now that the Bayesian framework was discussed, one can argue for a prior model on the noise standard deviation σ corresponding to observation and model errors. Usually, a Gamma prior with parameters a, b is adopted for the precision σ^{-2}. With this choice, the standard deviation σ can be analytically integrated out from the global likelihood (3.2), thus leading to a new global likelihood expression, taking the form of a Student's t-distribution:

$$f\Big(\boldsymbol{y}^{\text{obs}}|\boldsymbol{\theta}\Big) = \frac{\Gamma(a+\frac{M}{2})b^a}{\Gamma(a)(2\pi)^{\frac{M}{2}}}\left(b+\frac{1}{2}\sum_{i,j}\left\|\boldsymbol{u}_{i,j}^{\text{obs}}-\boldsymbol{u}_{i,j}^{\text{FE}}(\boldsymbol{\theta})\right\|^2\right)^{-(a+\frac{M}{2})}. \tag{3.4}$$

This expression of the global likelihood (3.4) is to be used instead of (3.2) when the noise standard deviation σ is unknown, relating to the use of the Student's t-distribution for robust inference in applied statistics.

The systems of interest are considered complex in the sense that the posterior distribution $\pi\Big(\boldsymbol{\theta}|\boldsymbol{y}^{\text{obs}}\Big)$ is not known explicitly, as it can only be evaluated for a particular $\boldsymbol{\theta}$, up to a constant multiplicative factor, with a call to a potentially costly deterministic forward solver (*e.g.* finite element codes). The number of calls to such a solver, via the likelihood evaluations, dominates the total computational cost, so that brute force recovery of the posterior is infeasible. For this reason, we adopt a sampling strategy in which samples of the posterior distribution are produced with advanced Monte Carlo strategies, also eliminating the need for a potentially high dimensional integration over the parameter space to determine the posterior normalizing term. Traditionally Markov chain Monte Carlo (MCMC) methods have been employed [49, 109], based on building a Markov chain with the posterior as its equilibrium distribution by defining an appropriate transition kernel [8, 83, 68, 48]. However, in cases where the posterior density has multiple modes, very large mixing times might be required and the rate of convergence can be extremely slow, thus leading to many calls to the forward solver. For these reasons we propose using Sequential Monte Carlo schemes [28, 29],

combined with the use of online tracking and interpolation of reduced order models [6, 5], to reduce the overall computational cost related to repeated likelihood evaluations.

3.3 Solution

3.3.1 Adaptive Sequential Monte Carlo

Sequential Monte Carlo schemes are flexible simulation-based methods for sampling from a sequence of probability distributions [20, 31, 63], using a set of random samples (referred as *particles*) which are propagated using a combination of importance sampling, resampling, and MCMC-based rejuvenation mechanisms [28, 29]. We point out that, as with Markov chain Monte Carlo schemes, the probability distributions need to be known only up to a constant. All particles are associated with respective *importance weights*, updated sequentially with the particle locations. If $\{\boldsymbol{\theta}^{(n)}, W^{(n)}\}_{n=1..N}$ represent N such particles and associated normalized weights (*i.e.* $\sum_{n=1}^{N} W^{(n)} = 1$) for the target probability distribution $\pi(\boldsymbol{\theta}|\boldsymbol{y}^{\text{obs}})$ then

$$\pi\Big(\boldsymbol{\theta}|\boldsymbol{y}^{\text{obs}}\Big) \approx \sum_{n=1}^{N} W^{(n)}\, \delta_{\boldsymbol{\theta}^{(n)}}(\boldsymbol{\theta})$$

where $\delta_{\boldsymbol{\theta}^{(n)}}(\cdot)$ is the Dirac delta function centered at $\boldsymbol{\theta}^{(n)}$. Furthermore, for any π-integrable function $h(\boldsymbol{\theta})$ we have the following convergence result [24, 27]:

$$\sum_{n=1}^{N} W^{(n)}\, h\Big(\boldsymbol{\theta}^{(n)}\Big) \xrightarrow{\text{a.s.}} \int h(\boldsymbol{\theta})\pi\Big(\boldsymbol{\theta}|\boldsymbol{y}^{\text{obs}}\Big)\,\mathrm{d}\boldsymbol{\theta}.$$

The main idea behind SMC algorithms is to operate on a sequence of distributions, starting from one that can be accurately and easily sampled from, and gradually moving towards the target density. Similar to simulated annealing, we consider the following sequence of intermediate distributions parameterized by $\gamma \in [0, 1]$, playing the role of reciprocal temperature:

$$\pi_{\gamma}\Big(\boldsymbol{\theta}|\boldsymbol{y}^{\text{obs}}\Big) \propto f\Big(\boldsymbol{y}^{\text{obs}}|\boldsymbol{\theta}\Big)^{\gamma}\pi(\boldsymbol{\theta}). \tag{3.5}$$

It is easily seen that the prior distribution $\pi(\boldsymbol{\theta})$ is recovered for $\gamma = 0$ and the posterior distribution is recovered for $\gamma = 1$. Increasing γ can be interpreted as progressively bringing in information via the likelihood $f\left(\boldsymbol{y}^{\text{obs}}|\boldsymbol{\theta}\right)$. Starting at $\gamma = 0$, we draw samples directly from $\pi(\boldsymbol{\theta})$ and set the weights $W^{(n)} = 1/N$; the goal being to gradually update the weights and the particle locations, in order to approximate the target posterior density. The intermediate distributions π_γ for $\gamma \in (0, 1)$ provide a smooth transition path where importance sampling can be efficiently applied.

As the number of intermediate distributions is increased, we can expect an increased accuracy since the transition becomes smoother, but the computational cost also increases as more evaluations of the likelihood would be needed. On the other hand, too few intermediate distributions can adversely affect the overall accuracy of the particulate approximation. To that end we propose an adaptive SMC scheme used in uncertainty quantification, stochastic design and system identification applications [57, 58, 104], which determines automatically the number of needed intermediate distributions based on the effective sample size. The effective sample size is defined as $\text{ESS} = 1/\sum_{n=1}^{N}(W^{(n)})^2 \in [1, N]$ and provides a measure of the variance of the importance weights [63]. Consequently, we define an acceptable reduction of the ESS from one distribution $\pi_{\gamma_{s-1}}$ to the next one π_{γ_s}, such that $\text{ESS}(\gamma_s) \geqslant \zeta\,\text{ESS}(\gamma_{s-1})$, with $\zeta < 1$, and then π_{γ_s} can be prescribed accordingly. Resampling the particles avoids a potential degeneracy of the algorithm when the ESS falls to low values; the extreme case being all the mass concentrated on a single particle of unit normalized weight, leading to $\text{ESS} = 1$. Such a situation is prevented by resampling the particles with a probability corresponding to their weights, when some criterion is reached: usually when the effective sample size reaches $N/2$, half the number of particles. Then the importance weights are reset to $1/N$ to get $\text{ESS} = N$. The details of the algorithm, as implemented in this work, appear in Algorithm 2.

Algorithm 2: Adaptive SMC sampler

Initialization: Set $s = 1$, $\gamma_s = 0$, for all $n \in \{1, \dots, N\}$ draw $\boldsymbol{\theta}_1^{(n)} \sim \pi(\boldsymbol{\theta})$ and set all weights $\{W_1^{(n)}\}_{n=1..N}$ to $1/N$.

while $\gamma_s \neq 1$ **do**

Set $s \leftarrow s + 1$

Weights updating

Compute the unnormalized weights for each particle $n \in \{1, \dots, N\}$

$$w_s^{(n)}(\gamma_s) = W_{s-1}^{(n)} \frac{\pi_{\gamma_s}\left(\boldsymbol{\theta}_{s-1}^{(n)} | \boldsymbol{y}^{\text{obs}}\right)}{\pi_{\gamma_{s-1}}\left(\boldsymbol{\theta}_{s-1}^{(n)} | \boldsymbol{y}^{\text{obs}}\right)} = W_{s-1}^{(n)} f\left(\boldsymbol{y}^{\text{obs}} | \boldsymbol{\theta}_{s-1}^{(n)}\right)^{\gamma_s - \gamma_{s-1}}$$

where γ_s is found such that $\text{ESS}(\gamma_s) = \zeta \, \text{ESS}(\gamma_{s-1})$ ($\zeta = 0.95$ in the examples). Then normalize the weights to obtain $\{W_s^{(n)}\}_{n=1..N}$.

Resampling

Compute the effective sample size $\text{ESS}(\gamma_s) = 1/\sum_{n=1}^{N} (W_s^{(n)})^2$ and resample if $\text{ESS}(\gamma_s) < N/2$.

Rejuvenating

Perturbate the particles with a Metropolis-Hastings kernel $K_s(\cdot, \cdot)$ of invariant distribution $\pi_{\gamma_s}(\boldsymbol{\theta} | \boldsymbol{y}^{\text{obs}})$, for $n \in \{1, \dots, N\}$ draw $\boldsymbol{\theta}_s^{(n)} \sim K_s(\boldsymbol{\theta}_{s-1}^{(n)}, \cdot)$.

end while

It should be noted that unlike MCMC schemes, the particle perturbations in the rejuvenating step do not require the π_{γ_s}-invariant Metropolis-Hastings kernel $\mathcal{K}_s(\cdot, \cdot)$ to be ergodic [28], thus allowing adaptive change of its parameters to achieve better mixing rates. Various MCMC kernels will be discussed in section 3.4.

Finally, we point out that the proposed adaptive SMC algorithm is embarrassingly parallellizable, as the analysis for each particle can be performed independently during the

rejuvenation step. An additional advantage of the proposed approach is that the algorithm outputs an empirical distribution approximating the posterior distribution, thus enabling its direct use as a prior distribution for subsequent data, if the latter is collected sequentially, as in real time system identification problems.

3.3.2 Reduced-order model tracking and interpolation

In most problems of interest, the major part of the total computational cost resides in the rejuvenating step of Algorithm 2, as repeated evaluations of the likelihood are needed to compute Metropolis-Hastings accept-reject ratios; each one implying a run of the forward solver. Although we are using an efficient SMC sampling scheme, the computational cost can become tremendously expensive as large linear systems, resulting from the discretization of the governing PDEs, need to be solved for each run.

We propose here to use projection based reduced order models (ROMs), represented by their $n \times r$ orthogonal projection matrix $\mathbf{R}$, where n is the dimension of the full-order system and r is the dimension of the approximation subspace. Choices of reduced order modeling techniques include the reduced basis method [71, 79, 80] and proper orthogonal decomposition (POD, [50, 102] and references therein) or principal components analysis (PCA, [76, 51]). Such a projection matrix $\mathbf{R}$ is typically built using selected response snapshots from the full order model, taken at different times t and/or for different values of the parameters $\boldsymbol{\theta}$. The present work adopts a POD approach to build those projection matrices. We consider a collection of m different responses from the full order model as an ensemble of snapshots; where each snapshot corresponds to a n-dimensional output vector, resulting in a $n \times m$

snapshot matrix $\mathbf{U}$, generally centered by subtracting the mean response to each snapshot:

$$\mathbf{U} = \begin{bmatrix} U_{11} & U_{12} & \cdots & U_{1m} \\ \vdots & \vdots & \ddots & \vdots \\ U_{n1} & U_{n2} & \cdots & U_{nm} \end{bmatrix}.$$

The POD modes are the eigenvectors of the covariance matrix $\mathbf{C} = \mathbf{U}\mathbf{U}^T$, or equivalently the left singular vectors of the snapshot matrix $\mathbf{U}$. The $n \times r$ orthogonal projection matrix $\mathbf{R}$ is then obtained from a subset of those modes corresponding to the r largest eigenvalues or singular values, so as to capture a large part of the total variance by observing the decay of the eigenvalues: in the numerical examples, we chose r such that on average 90% of the total variance is conserved.

We advocate here the use of one specific projection matrix $\mathbf{R}$ for each particular parameter vector $\boldsymbol{\theta}$, thus the POD modes are computed using only time history responses for a specific value of $\boldsymbol{\theta}$. This choice is motivated by the desire to improve the accuracy and reliability of the reduced order approach over the whole parameter space by avoiding a great loss of information about the system. Indeed, the POD projection matrix $\mathbf{R}$ contains, at most, as much information as the snapshots provide; thus leading to a poor reduced order approximation if the sought for solution is far from the subspace spanned by the columns of $\mathbf{R}$: for instance when localized phenomena are directly related to the parameter vector $\boldsymbol{\theta}$.

However, in our probabilistic setting described earlier, various and previously unseen model parameters are hypothesized during the sampling process, and so there is no reason to construct a projection matrix $\mathbf{R}$ using snapshots of the full order model for every new set of parameters $\boldsymbol{\theta}$; as it would be highly inefficient and likely never be needed again. We are therefore seeking a reduced order approach conserving as much information as possible, while at the same time having a controlled computational cost, enabling a fast solution to the inverse problem. In the context of the reduced basis method, a very efficient adaptation

procedure to solve parametrized partial differential equations was developed and analyzed in [87, 81], though restricted to a limited class of problems and using a single reduced basis matrix $\mathbf{R}$ to approximate the solutions over the parameter space.

The present work employs an interpolation approach [6], by taking a fixed population of projection matrices $\{\mathbf{R}_1, \ldots, \mathbf{R}_K\}$, built in an offline setting, associated with a specific set of parameters $\{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K\}$. We subsequently track and interpolate projection matrices along the path followed by the particles in the SMC scheme, through the consideration of the reference population; thus yielding an interpolated projection matrix $\mathbf{R}_*$, as needed to compute efficiently the likelihood for a new set of parameters $\boldsymbol{\theta}_*$, without a need for solving the full order system for these values of the parameters. However, interpolating reduced order models, or more generally orthogonal matrices or subspaces, is not a trivial task, since such entities do not belong to a linear space. For completeness, we expose in the following the interpolation procedure along with the necessary, more abstract theoretical considerations.

Firstly, it is pointed out that two $n \times r$ projection matrices $\mathbf{R}_1$ and $\mathbf{R}_2$ would lead to the same approximation of the solution of the full order model if there exists an $r \times r$ orthogonal matrix $\mathbf{Q}$ such that $\mathbf{R}_1 = \mathbf{R}_2\mathbf{Q}$. Indeed, their columns span the same r-dimensional subspace of $\mathbb{R}^n$, and since we consider projection based ROMs, the fundamental entity becomes the subspace in which the approximation is sought rather than one of its infinitely many bases which would all be equivalent. The approximation subspace $\mathcal{R}$ is thus represented as an element of the Grassmann manifold $\mathcal{G}_{n,r}$, namely the set of all r-dimensional subspaces of $\mathbb{R}^n$, and can be represented non-uniquely by an $n \times r$ projection matrix $\mathbf{R}$ belonging to the orthogonal Stiefel manifold $\mathcal{S}_{n,r}$, defined as the set of all $n \times r$ orthogonal matrices. Those spaces are obviously nonlinear, *i.e.* not "flat", however they possess a Riemannian structure enabling us, as we shall see, to perform the "interpolation" task. Indeed, the notion of distance between two points on a Riemannian manifold is intuitively interpreted through the

geodesic curves, having the property of minimizing the length of the path between points, as gauged by a Riemannian metric [30]. Furthermore, the additional quotient manifold structure of the Grassmann manifold $\mathcal{G}_{n,r}$ [36, 2], as a quotient of the orthogonal Stiefel manifold $\mathcal{S}_{n,r}$ by the orthogonal group $\mathcal{O}_r$, will help us in describing the currently adopted computational procedure, by considering elements of $\mathcal{G}_{n,r}$ while performing operations on elements of $\mathcal{S}_{n,r}$.

Since the interpolation cannot be easily performed directly in the Grassmann manifold $\mathcal{G}_{n,r}$, the proposed approach makes use of the bijective relation between end points of normalized geodesics on the manifold starting at a point $\mathcal{R}$, and elements of the tangent space $\mathcal{T}_{\mathcal{R}}\mathcal{G}_{n,r}$ of the Grassmann manifold at $\mathcal{R}$. This bijective relation is materialized through the *exponential map*, mapping elements from the tangent space to the manifold, and its reciprocal the *logarithm map*. Starting from a reference point $\mathcal{R}_0$, we map a subset S of the reference population $\{\mathcal{R}_1, \ldots, \mathcal{R}_K\}$ to the tangent space at $\mathcal{R}_0$, by way of the logarithm mapping. The tangent space being a vector space, classical multivariate interpolation can then be performed, and the result mapped back to the manifold with the exponential mapping (Figure 3.1); to get the desired point $\mathcal{R}_*$. Algorithm 3 summarizes the interpolation procedure [6], using matrix operations and formulas for the exponential and logarithm maps resulting from the aforementioned quotient structure [1, 10]; thus enabling a proper representation of elements $\mathcal{R}$ of the Grassmann manifold by elements $\mathbf{R}$ of the orthogonal Stiefel manifold, and analogously for elements of their respective tangent bundles, defined as the union of the tangent spaces.

In the example problems considered later, the subset S of the reference population is chosen as the $\{\mathcal{R}_s\}_{s\in S}$ such that the $\{\boldsymbol{\theta}_s\}_{s\in S}$ are appropriately selected neighbors of $\boldsymbol{\theta}_*$, and

Algorithm 3: Reduced-order model interpolation scheme

Input: Current ROM and associated parameter $\{\mathcal{R}_0, \boldsymbol{\theta}_0\}$, reference population $\{\mathcal{R}_k, \boldsymbol{\theta}_k\}_{k=1..K}$ and new parameter $\boldsymbol{\theta}_*$.

Step 1: Mapping to the tangent space $\mathcal{T}_{\mathcal{R}_0}\mathcal{G}_{n,r}$

Take a subset $\{\mathcal{R}_s, \boldsymbol{\theta}_s\}_{s\in S}$ of the reference population and compute their logarithm as follows:

$$\text{(Thin SVD)} \qquad (\mathbf{I} - \mathbf{R}_0\mathbf{R}_0^T)\,\mathbf{R}_s\,(\mathbf{R}_0^T\mathbf{R}_s)^{-1} = \mathbf{U}_s\boldsymbol{\Sigma}_s\mathbf{V}_s^T$$
$$\boldsymbol{\Gamma}_s = \log_{\mathcal{R}_0}\mathbf{R}_s = \mathbf{U}_s\tan^{-1}(\boldsymbol{\Sigma}_s)\mathbf{V}_s^T$$

where $\operatorname{span}(\mathbf{R}_0) = \mathcal{R}_0$ and $\operatorname{span}(\mathbf{R}_s) = \mathcal{R}_s$.

Step 2: Interpolation in the tangent space

Compute the interpolation weights $\{\alpha_s\}_{s\in S}$ to obtain the new tangent vector $\boldsymbol{\Gamma}_* = \sum_{s\in S}\alpha_s\boldsymbol{\Gamma}_s$.

Step 3: Mapping back to $\mathcal{G}_{n,r}$

Get the new ROM $\mathcal{R}_* = \operatorname{span}(\mathbf{R}_*)$ by computing the exponential of $\boldsymbol{\Gamma}_*$ as follows:

$$\text{(Thin SVD)} \qquad \boldsymbol{\Gamma}_* = \mathbf{U}_*\boldsymbol{\Sigma}_*\mathbf{V}_*^T$$
$$\mathbf{R}_* = \exp_{\mathcal{R}_0}\boldsymbol{\Gamma}_* = \mathbf{R}_0\mathbf{V}_*\cos\boldsymbol{\Sigma}_* + \mathbf{U}_*\sin\boldsymbol{\Sigma}_*.$$

the weights $\{\alpha_s\}_{s\in S}$ are simply given by

$$\tilde{\alpha}_s = \exp\left(-\frac{\|\boldsymbol{\theta}_s - \boldsymbol{\theta}_*\|}{\kappa}\right) \quad \text{and} \quad \alpha_s = \frac{\tilde{\alpha}_s}{\sum_{s\in S}\tilde{\alpha}_s}$$

where $\kappa > 0$ is a scaling parameter related to the sparsity of the reference population. Other less general, and more problem-specific, interpolation schemes can be readily chosen: for instance to take into account the various scales of variability of the different components

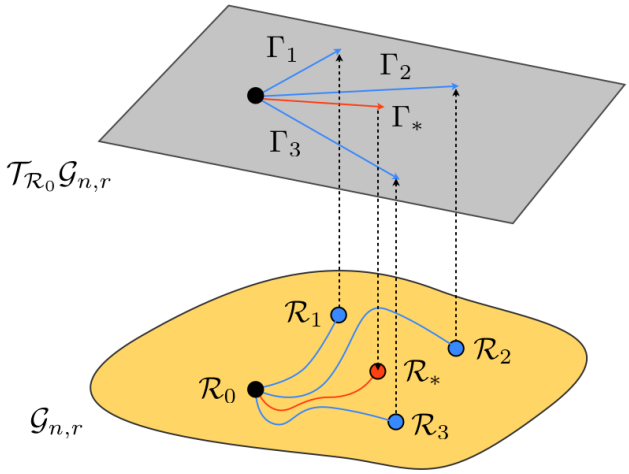


Figure 3.1: Interpolation procedure

of the parameter vector $\boldsymbol{\theta}$. As with any interpolation scheme, good performance is only achieved for lower parameter dimensions due to the curse of dimensionality. Improvement of this performance can be obtained with such specific interpolation schemes, as well as with a more insightful sampling of the initial population $\{\mathcal{R}_k, \boldsymbol{\theta}_k\}_{k=1..K}$, enabling a better representation of the parameter space for an equivalent computational cost.

3.3.3 Discussion of the inverse problem solution scheme

The interpolation scheme is integrated in the adaptive Sequential Monte Carlo scheme described in section 3.3.1 within the rejuvenating step, where interpolated approximation subspaces are uncovered, used to compute the likelihood, and updated while the particles are moved through the Metropolis-Hastings kernel. The current parameter value $\boldsymbol{\theta}_0$ is associated with the current reference approximation subspace $\mathcal{R}_0$ (represented by a projection matrix

$\mathbf{R}_0$), and with a proposal move to $\boldsymbol{\theta}_*$ an interpolated subspace $\mathcal{R}_*$ (represented by a projection matrix $\mathbf{R}_*$) is generated in order to compute accurately the accept-reject ratio. If the move to $\boldsymbol{\theta}_*$ is accepted, then the current reference approximation subspace is set to $\mathcal{R}_*$. Therefore, the reduced order models are "tracked" and interpolated as needed during the sampling process, to enable an accurate and efficient evaluation of the likelihood.

The computational cost of the interpolation scheme mainly stems from the singular value decomposition involved in the exponential and logarithm mappings and performed with $\mathrm{O}(nr^2)$ operations, to be compared with the $\mathrm{O}(n^2)$ or $\mathrm{O}(n^3)$ operations needed to solve a full order system; thus significant computational savings are achieved for large-scale problems exceeding some critical size. Indeed, the dependence upon the full order system dimension n for the interpolation of the reduced order model, and the assembly of the reduced system matrix during the sampling process, is the price to pay for the increased reliability and accuracy related to the use of a specific approximation subspace for a given parameter; compared to faster approaches working directly with reduced system matrices generated from a single reduced basis (*e.g.* [46, 26]).

3.4 Numerical examples

The two problems examined in this chapter are based on transient heat diffusion and governed by the following partial differential equation:

$$\frac{\partial u}{\partial t} = \nabla \cdot (k(\boldsymbol{x}) \nabla u) + f(\boldsymbol{x}, t) \ , \quad \boldsymbol{x} \in \Omega. \tag{3.6}$$

The conductivity field $k(\boldsymbol{x})$ and/or source term $f(\boldsymbol{x}, t)$ depend on the unknown parameter vector $\boldsymbol{\theta}$, and as a result the solution field $u(\boldsymbol{x}, t)$ will also implicitly depend on $\boldsymbol{\theta}$. The boundary conditions, the parameterization of the equation by the parameters $\boldsymbol{\theta}$ and its prior

specification are detailed in each of the specific examples below. The finite element method is employed for the spatial discretization of Equation (3.6), along with a backward finite difference scheme to approximate the time derivative, leading to the following system of algebraic equations for each time step:

$$\mathbf{A}_t(\boldsymbol{\theta})\boldsymbol{u}_t = \boldsymbol{f}_t(\boldsymbol{\theta}). \tag{3.7}$$

The solution field at time t is then recovered for a given $\boldsymbol{\theta}$ by solving the following reduced linear system of equations, using the ROM interpolation approach described in section 3.3.2:

$$\mathbf{B}_t(\boldsymbol{\theta})\boldsymbol{v}_t = \boldsymbol{g}_t(\boldsymbol{\theta}) \ , \quad \text{with } \mathbf{B}_t(\boldsymbol{\theta}) = \mathbf{R}^T\mathbf{A}_t(\boldsymbol{\theta})\mathbf{R} \text{ and } \boldsymbol{g}_t(\boldsymbol{\theta}) = \mathbf{R}^T\boldsymbol{f}_t(\boldsymbol{\theta}). \tag{3.8}$$

The approximation of $\boldsymbol{u}_t$ is then given by $\tilde{\boldsymbol{u}}_t = \mathbf{R}\boldsymbol{v}_t$, solving a $r \times r$ system instead of a $n \times n$ system. Note that to save computer memory, the reduced system of equations (3.8) could be assembled directly, following a Galerkin projection of the governing equation (3.6) onto the POD modes given by $\mathbf{R}$, thus avoiding the assembly and the storage of the full order system (3.7). Furthermore, additional savings can be made in the case of linear problems by computing the system matrices only once initially, whereas for nonlinear problems those matrices would have to be recomputed at each time step.

Before presenting the results, we provide some details on the MCMC kernels used in the rejuvenating step of the proposed adaptive SMC scheme described in Algorithm 2. We employ a Metropolis-within-Gibbs [83] or componentwise Hastings [14] sampler for the components of the parameter vector $\boldsymbol{\theta}$: the components $(\theta_1, \ldots, \theta_P)$ are sampled in separate groups from the full conditionals $\pi_{\gamma_s}\Big(\{\theta_p\}_{p\in Q}|\{\theta_p\}_{p\in Q^c}, \boldsymbol{y}^{\text{obs}}\Big)$, where $Q \uplus Q^c = \{1, \ldots, P\}$. A random walk Metropolis-Hastings scheme is used towards this goal, in which the standard deviation of the moves for a group of components is adaptively adjusted at each iteration of the SMC algorithm to ensure an acceptance rate between 20% and 30% from the accept-reject step. The idea is that the exploration of the parameter space would be performed

in a more efficient way, since sampling the components of the parameter vector in different groups of related variables would allow larger moves to be accepted.

3.4.1 Phase difference identification

The first numerical application consists in a moving source, parameterized by a phase variable which we aim to identify. We consider the following problem on the square $\Omega = [-1, 1]^2$ for $t \in [0, 1]$,

$$\begin{cases} \dfrac{\partial u}{\partial t} = \nabla \cdot (k \nabla u) + f(\boldsymbol{x}, t) & \text{on } \Omega \\ u(\boldsymbol{x}, 0) = u_0 & \text{on } \Omega \\ -k \frac{\partial u}{\partial n} = 0 & \text{on } \partial\Omega \end{cases} \tag{3.9}$$

in which the source term $f(\boldsymbol{x}, t)$ is given by:

$$f(\boldsymbol{x}, t) = A \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}_0(t)\|^2}{2\tau^2}\right) \quad \text{with} \quad \boldsymbol{x}_0(t) = \begin{pmatrix} \sin(2\pi t) \\ \sin(2\pi t + \phi) \end{pmatrix}.$$

In this example, the goal is to determine the phase shift parameter $\boldsymbol{\theta} = \phi \in [0, 2\pi]$. The numerical values $A = 10, \tau = 0.2, u_0 = 0$ were adopted for the other parameters. In terms of the POD approach, this is a nontrivial problem as it involves a moving and localized feature. It will be therefore enlightening to evaluate the performance of the utilized interpolation scheme in this setting, compared to using directly the full order forward model.

We adopt a uniform prior for $\boldsymbol{\theta} = \phi$ on the interval $[0, 2\pi]$, and we assume an *a priori* unknown variance for the noise in (3.1), so that the expression (3.4) is used for the likelihood. We choose $a = 0.01$ and $b = 0.01$ for the noise precision Gamma prior, corresponding to a diffuse distribution illustrating the lack of a priori information. Concerning the measurements, data were collected using a uniform array of 20 sensors at $t = 0.2, 0.4, 0.6, 0.8$. We generated the observed data $\boldsymbol{y} = \{u_{i,j}\}$ using a twice finer mesh (40×40 mesh, 1681 degrees

of freedom) than the one used in the analysis (20×20 mesh, 441 degrees of freedom) and an additive Gaussian noise with a signal over noise ratio of 50. The time step used during the backward finite difference time integration scheme was $\Delta t = 0.005$. The ground truth parameter value we employed is $\phi = 3\pi/2$.

The population of $K = 20$ POD projection matrices $\{\mathbf{R}_k, \boldsymbol{\theta}_k\}_{k=1..K}$ was computed uniformly on the interval $[0, 2\pi]$ at the points $\boldsymbol{\theta}_k = \phi_k = 2k\pi/K$, keeping $r = 10$ modes for each matrix. We ran the adaptive Sequential Monte Carlo scheme using $N = 200$ particles and 45 iterations, corresponding to 9,000 evaluations of the likelihood.

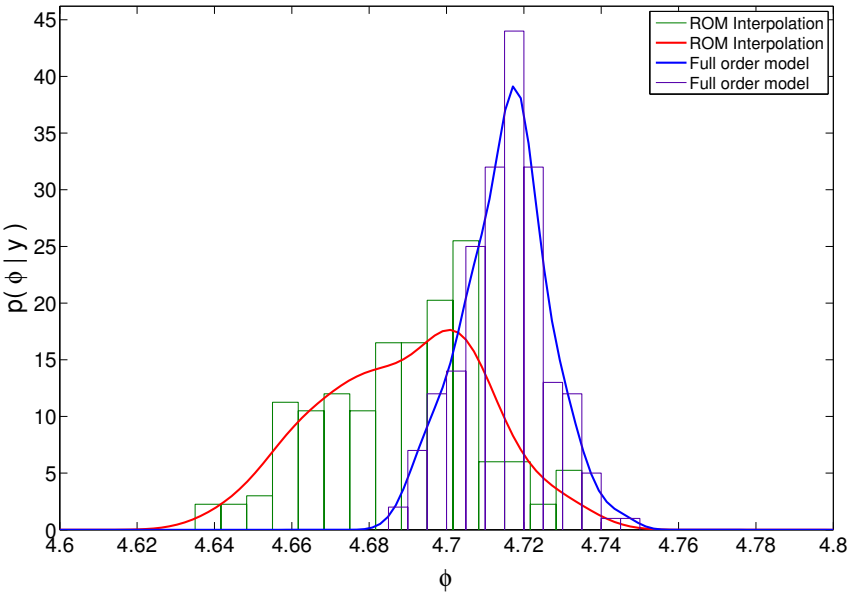


Figure 3.2: Histograms and kernel density estimates for the phase shift posterior distribution

Histograms and non-parametric kernel density estimates for the posterior distribution of the phase parameter ϕ are plotted in Figure 3.2. We also depicted a reference run using the same procedure but evaluating the likelihood directly with the full order forward model. We see that the proposed procedure succeeded in inferring the correct true parameter value

$\phi = 3\pi/2 = 4.712$; albeit with a small bias that we interpret as a consequence of the distribution of the POD projection matrices in $[0, 2\pi]$, the closest ones being at parameter values 4.62 and 4.96. As expected, the uncertainty about the true parameter for the reference run is smaller than with the proposed scheme, at the expense of an increased computational cost. We will investigate in the following numerical example the consequences of the sparsity of the initial population of POD projection matrices.

3.4.2 Source position identification

The second numerical application is a source identification problem, inspired from an example in [64]. We consider the following model in the square domain $\Omega = [-1, 1]^2$ for $t \in [0, 1]$,

$$\begin{cases} \dfrac{\partial u}{\partial t} = \nabla \cdot (k \nabla u) + f(\boldsymbol{x}, t) & \text{on } \Omega \\ u(\boldsymbol{x}, 0) = u_0 & \text{on } \Omega \\ -k \frac{\partial u}{\partial n} = 0 & \text{on } \Sigma_q \\ u(\boldsymbol{x}, t) = u_1(1 - \cos(2\pi\omega t)) & \text{on } \Sigma_u \end{cases} \tag{3.10}$$

in which the source term $f(\boldsymbol{x}, t)$ is given by

$$f(\boldsymbol{x}, t) = \frac{s e^{-\alpha t}}{2\pi\tau^2} \exp\left(- \frac{\|\boldsymbol{x} - \boldsymbol{x}^*\|^2}{2\tau^2} \right).$$

Homogeneous Neumann conditions are applied on $\Sigma_q = \{\{x_1 = 1\}, \{x_2 = -1\}, \{x_2 = 1\}\}$ and time-dependent Dirichlet conditions on $\Sigma_u = \{x_1 = -1\}$, see Figure 3.3. In our example, the parameter $\boldsymbol{\theta} = (x_1^*, x_2^*)$ corresponds to the position of the heat source, with $(x_1^*, x_2^*) \in [-1.25, 1.25]^2$ to allow for a source positioned slightly out of the domain Ω. The numerical values $k = 1, u_0 = 0, u_1 = 1, \omega = 5, s = 5, \tau = 0.2, \alpha = 1$ were adopted for the other parameters.

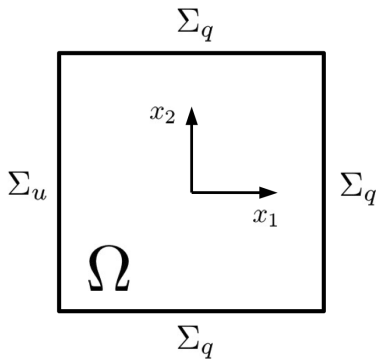


Figure 3.3: Domain and boundary conditions

A uniform prior was adopted in this study for $\boldsymbol{\theta} = (x_1^*, x_2^*)$ on the square $[-1.25, 1.25]^2$, and we also assumed an *a priori* unknown variance for the noise in (3.1), so that the expression (3.4) is used for the likelihood. The parameters $a = 0.01$ and $b = 0.01$ were chosen for the noise precision Gamma prior for the same reasons as in the previous application. In this example, the measurements were taken at three different times $t = 0.25, 0.5, 0.75$ using a uniform array of 20 sensors, thus leading to $M = 60$ measurements. The observed data $\boldsymbol{y} = \{u_{i,j}\}$ was synthetically generated from a twice finer mesh (40×40 mesh, 1681 degrees of freedom) than the one used to compute the population of POD projection matrices employed to compute the likelihood (20×20 mesh, 441 degrees of freedom); and $\boldsymbol{x}^* = (0, 0)$ was used for the ground truth parameter values. The backward finite difference time integration scheme used a time step $\Delta t = 0.01$. Gaussian noise was then added, characterized once again by a signal over noise ratio of 50.

Since the parameter space is now two-dimensional, the initial population of POD projection matrices was computed on a uniform grid for $\boldsymbol{\theta} \in [-1.25, 1.25]^2$, and $r = 10$ modes were kept to build those matrices. The adaptive Sequential Monte Carlo scheme described in section 3.3.1 was run with $N = 200$ particles. In addition to solving the inverse problem, we investigate here the influence of the "sparsity" of the precomputed population, $\{\mathbf{R}_k, \boldsymbol{\theta}_k\}_{k=1..K}$,

for grids with $K = 81$ points and $K = 25$ points respectively, on the posterior distribution.

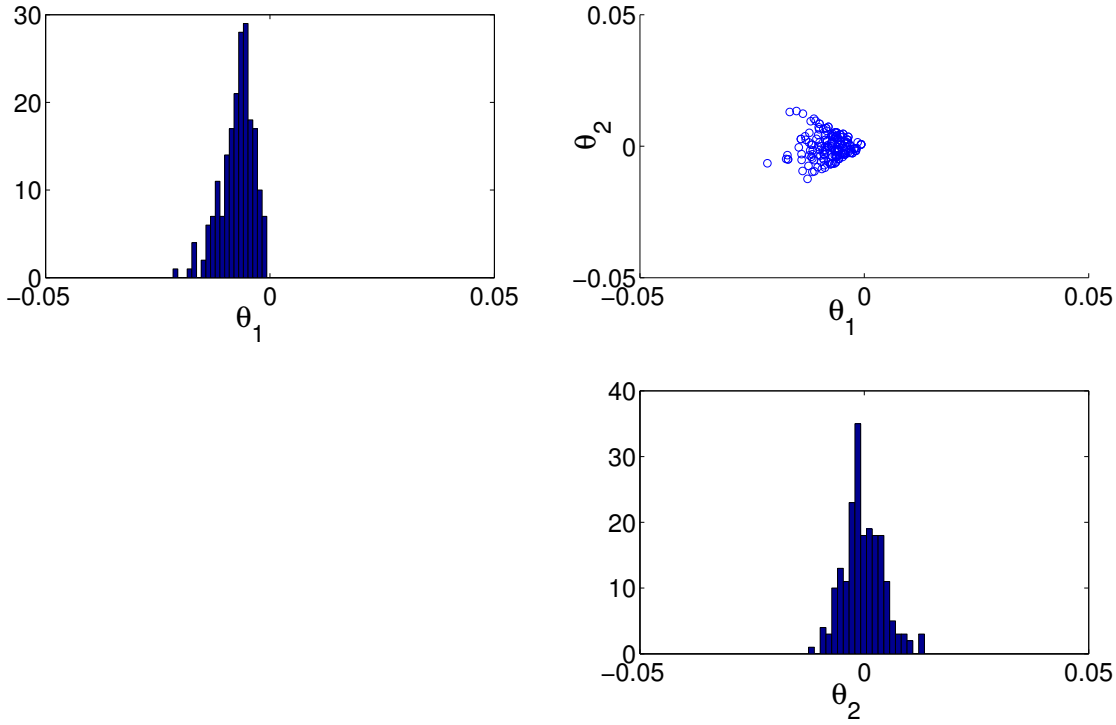


Figure 3.4: Histograms for $\theta_1 = x_1^*$ (top left), $\theta_2 = x_2^*$ (bottom right) and scatter plot θ_1 versus θ_2 (top right) for the problem (3.10) with $K = 81$ precomputed POD projection matrices

The results are presented in Figures 3.4-3.5, depicting the histograms of samples from the marginal posterior distributions of $\theta_1 = x_1^*$ and $\theta_2 = x_2^*$, as well as a scatter plot between θ_1 and θ_2. It can be seen that the proposed scheme correctly uncovered the true position of the source term $\boldsymbol{x}^* = (0, 0)$, albeit with a very small bias for θ_1. The posterior distribution exhibits a non-symmetrical shape, with more uncertainty in the region $\theta_1 < 0$. We interpret this behavior as a possible repercussion of the grid distribution of the initial population of POD projection matrices combined with the presence of a pronounced Dirichlet boundary condition on Σ_u. This behavior is shared for both $K = 81$ and $K = 25$, and as expected

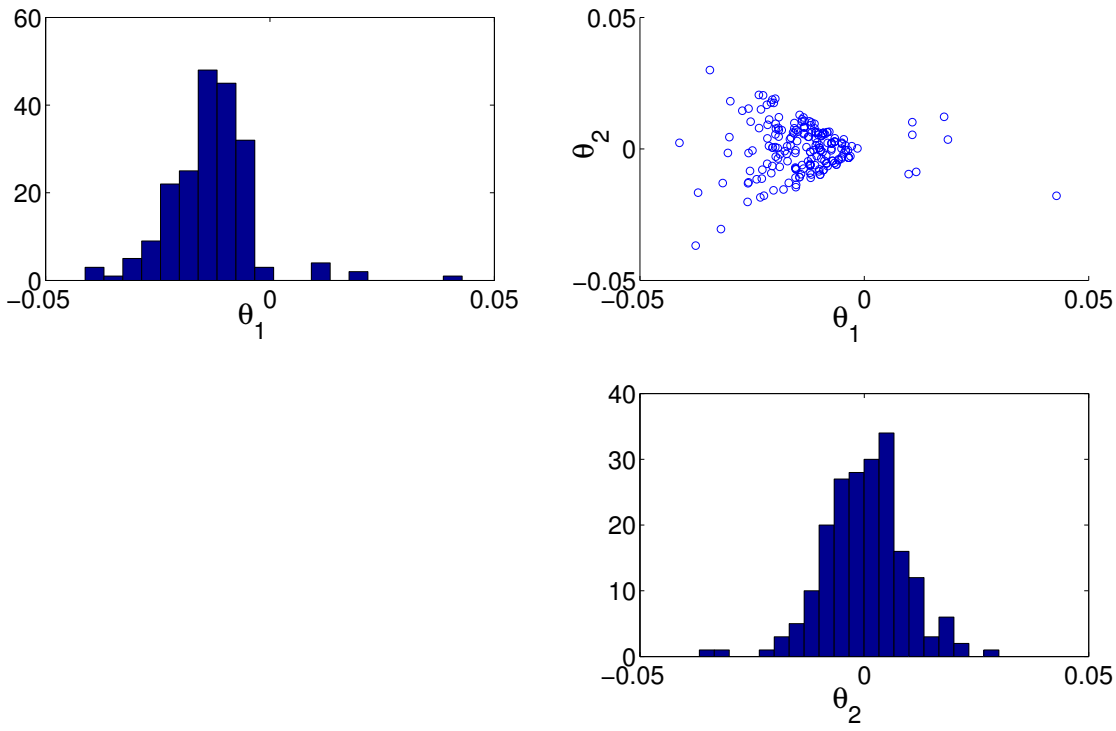


Figure 3.5: Histograms for $\theta_1 = x_1^*$ (top left), $\theta_2 = x_2^*$ (bottom right) and scatter plot θ_1 versus θ_2 (top right) for the problem (3.10) with $K = 25$ precomputed POD projection matrices

the uncertainty about the mode is greater in the latter case. For reference, the results using directly the full order model are plotted in Figure 3.6.

It is interesting to see that the use of a sparser reference database of projection matrices increases the model error. Indeed, the conditional posterior of the error variance from Equation (3.2), given the data and the parameters posterior mean, can be evaluated analytically as an Inverse-Gamma distribution, plotted in Figure 3.7. The observation noise being the same for both values of K, the latter figure shows that a finer reference population helps to decrease the prediction error, of which varying contributions are here the interpolation and projection errors.

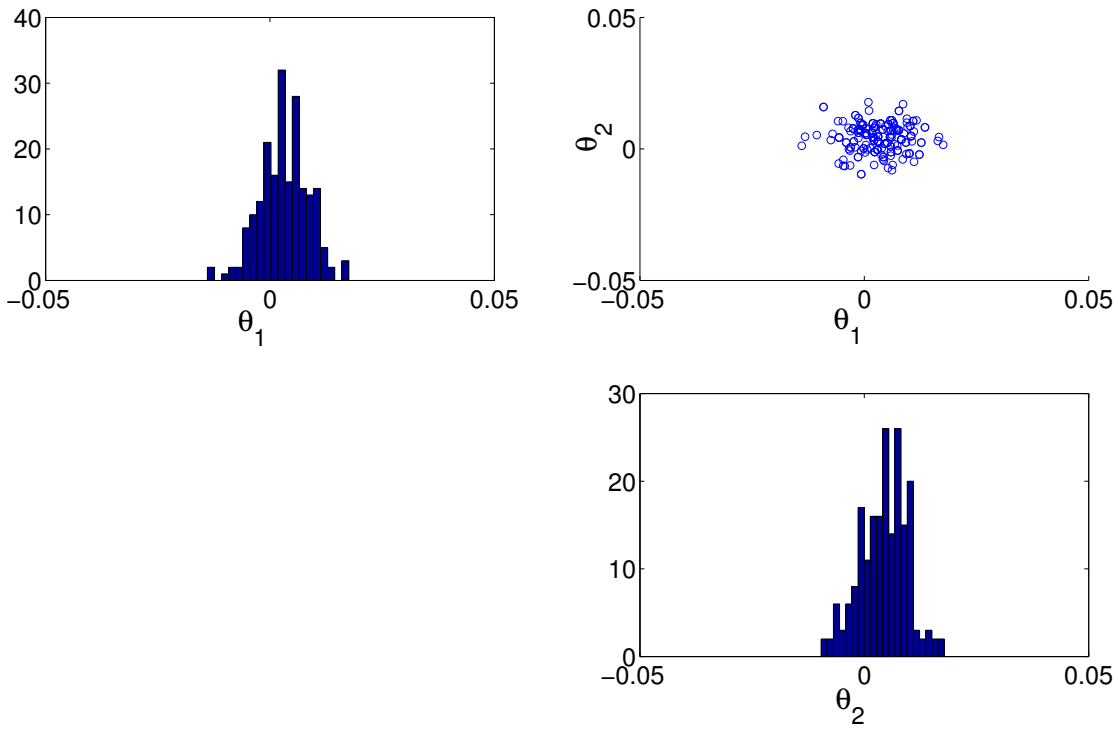


Figure 3.6: Histograms for $\theta_1 = x_1^*$ (top left), $\theta_2 = x_2^*$ (bottom right) and scatter plot θ_1 versus θ_2 (top right) for the problem (3.10) using the full order model

Concerning the computational cost, the SMC algorithm converged in 70 iterations, corresponding to 28,000 evaluations of the likelihood. This is comparable to what is used in practice with MCMC for similar problems, however we recall that SMC is embarrassingly parallelizable. Since the likelihood is evaluated using the reduced-order models interpolation scheme developed in section 3.3.2, significant computational savings are achieved compared to the direct use of the full order model; the latter being run only K times to build the initial population of POD projection matrices. Furthermore, the empirical distribution outputted by the proposed scheme can be directly used to evaluate the uncertainty about the parameter values, and if needed taken as the prior for a subsequent run, with an adaptively refined

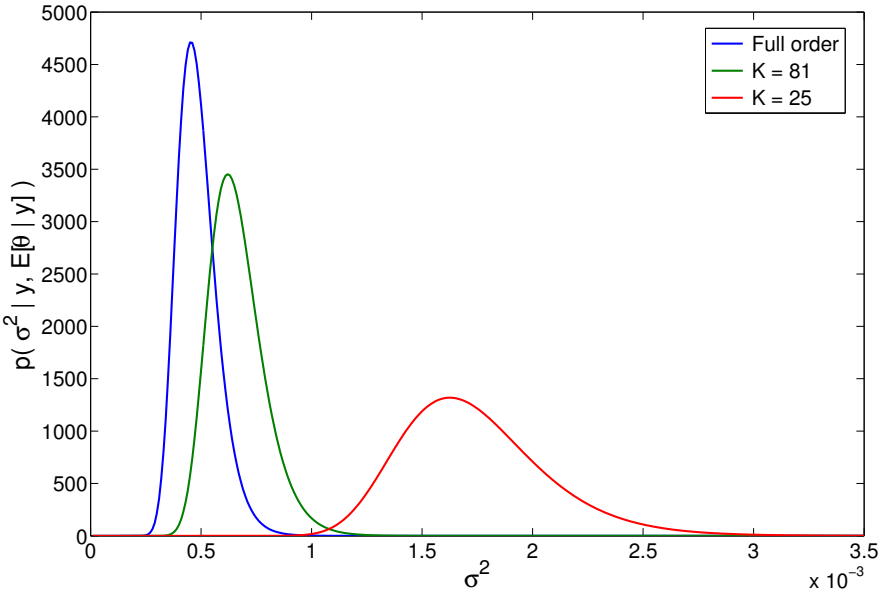


Figure 3.7: Conditional posterior distribution of the variance

database of projection matrices.

3.5 Conclusions

The present chapter has described a framework for effecting reduced order model tracking and interpolation as part of a Sequential Monte Carlo based sampling of joint posterior distributions. Such distributions arise naturally within uncertainty quantification contexts aimed at gauging the veracity and predictive power of partial differential equations postulated to describe physical systems. The example problems treated in this chapter demonstrate the promise of the proposed approach in furnishing significant computational savings while preserving sufficient fidelity to furnish a credible means for quantifying uncertainty regarding modeling parameters that are useful in describing physical systems.

CHAPTER 4

THE PARAMETER SPACE AS A RIEMANNIAN MANIFOLD: EFFECTIVE SAMPLING STRATEGIES FOR REDUCED ORDER MODELING

Abstract

This chapter presents a novel methodology to effectively sample relevant points in the parameter space, when collecting snapshots for reduced-order models from an expensive to evaluate computer model. Principles from *information geometry* are employed; where the parameter space is interpreted as a Riemannian manifold equipped with a sensitivity related metric. An adaptive Sequential Monte Carlo scheme enables an effective sampling of such points, while allowing a controlled computational cost, related to the evaluations of the computer model. Numerical examples demonstrating the proposed approach are provided.

4.1 Introduction

The Computational Science and Engineering (CSE) field has seen in recent years a drastic increase in the complexity of problems that can be solved as computer speed and resources have expanded. However, scientific interest in pushing frontiers even further has motivated computational scientists to develop more effective approaches to modeling. This has lead to the development of reduced order modeling methods, of which striking applications include leading edge design, uncertainty quantification and the solution of complex inverse problems [15, 75, 62, 38]. Indeed, for such applications, reduced order modeling (ROM) procedures prove themselves useful as a large number of repeated simulations of the system of interest

is required; each corresponding for instance to a set of relevant parameters to be optimized or inferred.

For systems being modeled by a set of partial differential equations, popular ROM methods have included the proper orthogonal decomposition (POD) method [50, 102] and the reduced basis (RB) method [71, 79, 80]. The main idea underpinning these methods is to form a linear system of equations much smaller than the one resulting from the direct discretization of the governing partial differential equations, by projecting the latter linear system onto a subspace related to a set of carefully selected system responses, called *snapshots*. Each snapshot corresponds to a particular set of modeling parameters describing a plausible modeling instance. The problem of snapshot selection then appears: given a certain number of snapshots to be computed, how does one sample the corresponding points in the model parameter space, to best represent the system response over the whole parameter space? A widely used approach is to take uniform samples, by using latin hypercube sampling or other low-discrepancy sequences. More sophisticated approaches have been developed in the past, sometimes using directly information about the system of interest. Important contributions have also been made related to the use of Centroidal Voronoi tesselations [34, 19] in reduced order modeling. Additionally, development of greedy sampling methods [46, 17] enabled significant progress in the snapshot selection problem, by exploiting heuristic arguments and optimization schemes to collect relevant sets of parameters.

We suggest here an alternative approach to model parameter selection that is based on geometric principles: the parameter space is interpreted as a Riemannian manifold, its metric emphasizing the important regions of the parameter space. The latter regions are the ones containing the most "information", *i.e.* the ones being the most sensitive in affecting the system response. The geometric reasoning employed here leads to a few connections with the field of statistical design of experiments [9, 21, 37, 88]. We use principles of *information*

geometry to guide our reasoning and a tensor interpolation procedure to build an approximation of the "information" metric, which cannot be computed practically over the whole parameter space. An adaptive *Sequential Monte Carlo* scheme and clustering techniques are then employed to sample effectively relevant parameters within the transformed parameter space, in view of generating snapshots.

The present chapter is organized as follows: in Section 4.2 we formulate our problem and provide a review of the geometric principles involved; then we discuss in Section 4.3 the chosen methodology, which includes the metric field approximation and the sampling framework. Two examples illustrating our approach in the context of the POD method are developed in Section 4.4, and subsequent conclusions are drawn in Section 4.5.

4.2 Problem description

4.2.1 Snapshot selection in reduced-order modeling

We consider a general system whose response $\boldsymbol{u}(\boldsymbol{\theta}) \in \mathbb{R}^n$ depends smoothly on p different parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_p) \in \boldsymbol{\Theta} \subset \mathbb{R}^p$. For instance, such response could be a discretized solution field of a system of partial differential equations governing the system of interest.

Our goal is, given a prescribed number m of calls to our computational model (due to limitations on resources or time), to come up with the best set of m parameters $\{\boldsymbol{\theta}\}_{k=1}^m$, such that the *snapshots* $\{\boldsymbol{u}^k = \boldsymbol{u}(\boldsymbol{\theta}_k)\}_{k=1}^m$ represent "best" the system response for the whole parameter space $\boldsymbol{\Theta}$. Those snapshots can then be used to build a reduced order model of the system of interest, using for instance the reduced basis method or the proper orthogonal decomposition (POD) method. Such reduced order models subsequently facilitate

the fast evaluation of the system response in the context of numerical methods based on the discretization of partial differential equations. For these methods, the approximate response obtained from the reduced order model belongs to the subspace span $\{\boldsymbol{u}^k\}_{k=1}^m$ of $\mathbb{R}^n$, hence the importance of a careful snapshots selection to enable better accuracy of the ROM.

4.2.2 Riemannian manifolds and Information geometry

Our approach is to interpret the parameter space $\boldsymbol{\Theta}$ as a Riemannian manifold: a curved space in which the Euclidean metric is replaced by a Riemannian metric dependent on $\boldsymbol{\theta}$. The idea is to devise a suitable metric that makes two parameters in an "important" (*i.e.* more sensitive) region of the parameter space $\boldsymbol{\Theta}$ "farther" by zooming in; and conversely in a less "important" (*i.e.* less sensitive) region of $\boldsymbol{\Theta}$ to make them "closer" by zooming out. The consequence is that uniform samples taken from the resulting curved space will naturally emphasize the "important" regions of the parameter space; which is of interest in our problem of snapshot selection: we obviously want to build a ROM using snapshots that involve the "important" parameter sets.

We will next first give a broad overview of the differential geometric tools that are needed, and then we will put the aforementioned general description into the practical context emanating from *information geometry*.

Riemannian geometry

A *differentiable manifold* M is a "smooth" set with a $\mathcal{C}^\infty$ complete collection of systems of coordinates defined on a collection of open sets comprising M. Simple examples of differentiable manifolds include the Euclidean space $\mathbb{R}^n$ for $n \in \mathbb{N}$, or the unit sphere

$S_n = \{\boldsymbol{x} \in \mathbb{R}^n \,|\, \|\boldsymbol{x}\|_2 = 1\}$. The tangent space $T_{\boldsymbol{x}}M$ is the set of all tangent vectors of M at a point $\boldsymbol{x} \in M$, and is a linear vector space.

The following definition is adapted from [30]. A *Riemannian metric* on a differentiable manifold M is a correspondence which associates to each point $\boldsymbol{x}$ of the manifold M an inner product $(\cdot,\cdot)_{\boldsymbol{x}}$ on its tangent space $T_{\boldsymbol{x}}M$, which varies smoothly. A differentiable manifold equipped with a Riemannian metric is called a *Riemannian manifold*. For a given system of coordinates $(x_1, \ldots, x_n)$ in a neighborhood $U \subset M$ of $\boldsymbol{x}_0 \in M$, the Riemannian metric can be represented as a matrix $\mathbf{G}_{\boldsymbol{x}}$ so that for two tangent vectors $\boldsymbol{\xi}_1, \boldsymbol{\xi}_2$ we can write $(\boldsymbol{\xi}_1, \boldsymbol{\xi}_2)_x = \boldsymbol{\xi}_1^T \mathbf{G}_{\boldsymbol{x}} \boldsymbol{\xi}_2$.

Information geometry and application to our problem

Information geometry [4] is a branch of statistics in which the parameter space of a given statistical model is interpreted as a Riemannian manifold, with the Riemannian metric furnished by the *Fisher information matrix*. Given a parameterized probability density function $p(\boldsymbol{y}|\boldsymbol{\theta})$ for $\boldsymbol{\theta} \in \boldsymbol{\Theta}$, the Fisher information matrix is given by:

$$\mathbf{I}(\boldsymbol{\theta}) = \mathrm{E}\left[\left(\frac{\partial}{\partial \boldsymbol{\theta}} \ln p(\boldsymbol{y}|\boldsymbol{\theta})\right)\left(\frac{\partial}{\partial \boldsymbol{\theta}} \ln p(\boldsymbol{y}|\boldsymbol{\theta})\right)^T \Big| \boldsymbol{\theta}\right] = -\mathrm{E}\left[\left(\frac{\partial^2}{\partial \boldsymbol{\theta}^2} \ln p(\boldsymbol{y}|\boldsymbol{\theta})\right)\right], \tag{4.1}$$

where $\mathrm{E}[\cdot]$ denotes here the expectation operator. Hence, the (squared) distance between two probability density functions $p(\boldsymbol{y}|\boldsymbol{\theta})$ and $p(\boldsymbol{y}|\boldsymbol{\theta}+\delta\boldsymbol{\theta})$ can be computed by $\delta\boldsymbol{\theta}^T \mathbf{I}(\boldsymbol{\theta}) \delta\boldsymbol{\theta}$. Therefore, the Fisher information, which is a symmetric positive (semi)definite matrix, provides a natural Riemannian structure on the parameter space $\boldsymbol{\Theta}$.

A canonical example is given by the normal distribution $y|\boldsymbol{\theta} \sim \mathcal{N}(\mu, \sigma^2)$ with $\boldsymbol{\theta} = (\mu, \sigma)$ and $\boldsymbol{\Theta}$ being the upper half-plane [44]. In this setting, the Fisher information matrix is given

by:

$$\mathbf{I}(\boldsymbol{\theta}) = \begin{bmatrix} 1/\sigma^2 & 0 \\ 0 & 2/\sigma^2 \end{bmatrix}.$$

We can compute the distance between (μ, σ) and $(\mu + \delta\mu, \sigma + \delta\sigma)$ as $(\delta\mu^2 + 2\delta\sigma^2)/\sigma^2$, which decreases as σ increases. This confirms the intuitive picture that for two given means, the resulting normal distributions are intuitively more different when the standard deviation is smaller.

In the following, we will now assume that $\boldsymbol{y}|\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{h}(\boldsymbol{u}(\boldsymbol{\theta})), \sigma^2 \boldsymbol{I})$ is the "observed" response of the system of interest, and we will see that in fact the value of the standard deviation σ does not matter for our purposes. The function $\boldsymbol{h}$ is here a smooth function of $\boldsymbol{u}(\boldsymbol{\theta})$, which itself depends smoothly on $\boldsymbol{\theta} \in \boldsymbol{\Theta}$. First, let us give the expression of the Fisher information matrix for our problem:

$$\mathbf{I}(\boldsymbol{\theta}) = \sigma^{-2} \left(\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{\theta}} \right)^T \left(\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{\theta}} \right). \tag{4.2}$$

At a point $\boldsymbol{\theta}$, the Riemannian metric given by $\mathbf{I}(\boldsymbol{\theta})$ induces an infinitesimal volume element on the tangent space $T_{\boldsymbol{\theta}}\boldsymbol{\Theta}$, which can be identified with $\boldsymbol{\Theta}$ in our case:

$$\mathrm{d}\mathcal{V}(\boldsymbol{\theta}) = \sqrt{\det \mathbf{I}(\boldsymbol{\theta})}\, \mathrm{d}\boldsymbol{\theta}.$$

The previous expression also gives a Riemannian measure on the manifold $\boldsymbol{\Theta}$ [77], and for the Lebesgue measure the uniform distribution can therefore be written as

$$p(\boldsymbol{\theta}) \propto \sqrt{\det \mathbf{I}(\boldsymbol{\theta})}, \tag{4.3}$$

which does not depend on the standard deviation σ, as was claimed above. Therefore, the previous formula provides the uniform distribution in the parameter space, interpreted as a Riemannian manifold equipped with the Fisher information metric; and corresponding samples would therefore be more concentrated in regions furnishing more information, *i.e.*

where the response is more sensitive with regard to the parameters. We can see that the expression (4.3) is related to the *D-optimality* criterion in the statistical design of experiments, described in [21, 37]. However, when one wishes to produce such samples, the following issues arise and will be the object of the subsequent sections:

1. The metric is a tensor field on the parameter space, given by the Fisher information matrix (4.2). As such, it is impracticable to evaluate it for each parameter since the required sensitivities entail calls to a computationally expensive model. We propose a strategy based on an interpolation approach that enables an approximate representation of the metric field.
2. Sampling arbitrary probability distributions, including (4.3), is a difficult problem. Markov chain Monte Carlo methods have been proposed in the past, however such schemes produce samples which can become trapped in local modes. Full exploration of the parameter space and non correlated samples are critical in our case; thus we propose the use of an adaptive Sequential Monte Carlo scheme.
3. It is well known that uniform samples in the Euclidean space are not optimal in terms of discrepancy; thus explaining the poor performance of uniform sampling with regard to our problem of model parameter selection, as compared to latin hypercube sampling or others quasi-random low discrepancy sequences. We propose the use of clustering algorithms to alleviate this issue once in the Riemannian space.

4.3 Methodology

4.3.1 Approximating the metric tensor field

In this section, we are describing an approximation strategy to alleviate the first issue mentioned earlier, namely computing the Fisher information matrix (4.2). Indeed, computing derivatives of the response, as needed during the sampling process, is prohibitively expensive for the complex systems of which we aim to furnish effective reduced-order models. A first solution would be to compute those derivatives using surrogate models, such as ones emanating from a coarser discretization in finite element schemes. However, this is often impractical, as a coarser discretization can fail to capture salient features of a given system, and the computational savings might be not as great as needed.

We suggest here an interpolation strategy, based on the work [78], in order to approximate the metric tensor field $\mathbf{I}(\boldsymbol{\theta})$. The idea is to precompute a fixed population of metric tensors $\{\mathbf{I}_1, \ldots, \mathbf{I}_K\}$ associated with a specific set of parameters $\{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K\}$. The required sensitivities can then be computed via finite difference schemes, or better yet, automatic differentiation procedures, as well as adjoint methods if the output of interest $\boldsymbol{h}(\boldsymbol{u}(\boldsymbol{\theta}))$ is a scalar. Fisher information matrices corresponding to arbitrary parameters are subsequently "interpolated", as needed during the sampling process, through the consideration of the reference population $\{\mathbf{I}_1, \ldots, \mathbf{I}_K\}$. The resulting interpolated metric tensor $\mathbf{I}_*$ enables an efficient computation of (4.3) for a new parameter $\boldsymbol{\theta}_*$, without any call to the forward model.

However, "interpolating" symmetric positive definite matrices is not a trivial task, since such entities do not belong to a linear space. It can be shown that the set of $p \times p$ symmetric positive definite matrices $\mathcal{S}_p$ can be equipped with a Riemannian structure [78]; thus enabling

us to perform the interpolation task. The proposed approach makes use of the bijective relation between end points of normalized geodesics on the manifold $\mathcal{S}_p$ starting at a point $\mathbf{I}_0$, and elements of the tangent space $\mathcal{T}_{\mathbf{I}_0}\mathcal{S}_p$. This bijective relation is materialized through the *exponential map*, mapping elements from the tangent space to the manifold, and its reciprocal the *logarithm map*. Starting from a reference point $\mathbf{I}_0$, we map a subset $\{\mathbf{I}_q\}_{q\in Q}$ of the reference population $\{\mathbf{I}_1, \ldots, \mathbf{I}_K\}$ to the tangent space at $\mathbf{I}_0$, by way of the logarithm map. The tangent space $\mathcal{T}_{\mathbf{I}_0}\mathcal{S}_p$ being a vector space, classical multivariate interpolation can then be performed; and the result mapped back to the manifold through the exponential map to get the desired point $\mathbf{I}_*$. Algorithm 4 and Figure 4.1 summarize the interpolation procedure, using matrix operations and formulas from [78] for the exponential and logarithm maps.

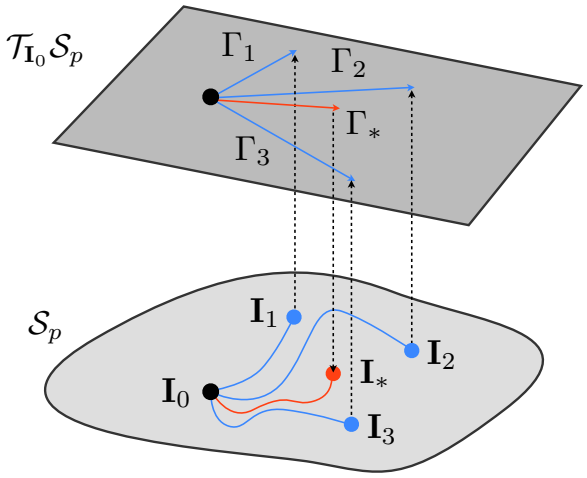


Figure 4.1: Interpolation procedure

In the example problems considered later, the subset $\{\mathbf{I}_q\}_{q\in Q}$ of the reference population corresponds to parameters $\{\boldsymbol{\theta}_q\}_{q\in Q}$ that are appropriately selected neighbors of $\boldsymbol{\theta}_*$; and $\eta > 0$ in the interpolation weights is a scaling parameter related to the sparsity of the reference

population.

Algorithm 4: Fisher information metric interpolation scheme

Input: Current FIM and associated parameter $\{\mathbf{I}_0, \boldsymbol{\theta}_0\}$, reference population $\{\mathbf{I}_k, \boldsymbol{\theta}_k\}_{k=1..K}$ and new parameter $\boldsymbol{\theta}_*$.

Step 1: Mapping to the tangent space $\mathcal{T}_{\mathbf{I}_0}\mathcal{S}_p$

Take a subset $\{\mathbf{I}_q, \boldsymbol{\theta}_q\}_{q\in Q}$ of the reference population and compute their logarithm as follows:

$$\boldsymbol{\Gamma}_q = \log_{\mathbf{I}_0} \mathbf{I}_q = \mathbf{I}_0^{\frac{1}{2}} \log\left(\mathbf{I}_0^{-\frac{1}{2}} \mathbf{I}_q \mathbf{I}_0^{-\frac{1}{2}}\right) \mathbf{I}_0^{\frac{1}{2}}$$

where the Γ_q are the images of the $\mathbf{I}_q$ from the FIM manifold $\mathcal{S}_p$, as realizations in the tangent space $\mathcal{T}_{\mathbf{I}_0}\mathcal{S}_p$.

Step 2: Interpolation in the tangent space

Compute the interpolation weights $\{\alpha_q\}_{q\in Q}$,

$$\alpha_q = \frac{\tilde{\alpha}_q}{\sum_{q\in Q} \tilde{\alpha}_q} \quad \text{with} \quad \tilde{\alpha}_q = \exp\Big(-\frac{\|\boldsymbol{\theta}_q - \boldsymbol{\theta}_*\|}{\eta}\Big)$$

to obtain the new tangent vector $\boldsymbol{\Gamma}_* = \sum_{q\in Q} \alpha_q \boldsymbol{\Gamma}_q$.

Step 3: Mapping back to $\mathcal{S}_p$

Obtain the new FIM $\mathbf{I}_*$ by computing the exponential of $\boldsymbol{\Gamma}_*$ as follows:

$$\mathbf{I}_* = \exp_{\mathbf{I}_0} \boldsymbol{\Gamma}_* = \mathbf{I}_0^{\frac{1}{2}} \exp\left(\mathbf{I}_0^{-\frac{1}{2}} \boldsymbol{\Gamma}_* \mathbf{I}_0^{-\frac{1}{2}}\right) \mathbf{I}_0^{\frac{1}{2}}$$

4.3.2 Sampling with Sequential Monte Carlo

Sequential Monte Carlo schemes are flexible simulation-based methods for sampling from a sequence of probability distributions [20, 23, 31, 63], using a set of random samples (referred

as *particles*) which are propagated using a combination of importance sampling, resampling, and MCMC-based rejuvenation mechanisms [28, 29]. We point out that, as with Markov chain Monte Carlo schemes [83, 68, 48], the probability distributions need to be known only up to a constant. All particles are associated with respective *importance weights*, updated sequentially with the particle locations. If $\{\boldsymbol{\theta}^{(n)}, W^{(n)}\}_{n=1..N}$ represent N such particles and associated normalized weights (*i.e.* $\sum_{n=1}^{N} W^{(n)} = 1$) for the target probability distribution $p(\boldsymbol{\theta})$ then

$$p(\boldsymbol{\theta}) \approx \sum_{n=1}^{N} W^{(n)}\, \delta_{\boldsymbol{\theta}^{(n)}}(\boldsymbol{\theta}) \tag{4.4}$$

where $\delta_{\boldsymbol{\theta}^{(n)}}(\cdot)$ is the Dirac delta function centered at $\boldsymbol{\theta}^{(n)}$. Furthermore, for any π-integrable function $h(\boldsymbol{\theta})$ we have the following convergence result [24, 27]:

$$\sum_{n=1}^{N} W^{(n)}\, h\Big(\boldsymbol{\theta}^{(n)}\Big) \xrightarrow{\text{a.s.}} \int h(\boldsymbol{\theta}) p(\boldsymbol{\theta})\, \mathrm{d}\boldsymbol{\theta}.$$

The main idea behind SMC algorithms is to operate on a sequence of distributions, starting from one that can be accurately and easily sampled from, and gradually moving towards the target density. Similar to simulated annealing, we consider the following sequence of intermediate distributions parameterized by $\gamma \in [0, 1]$, playing the role of reciprocal temperature:

$$p_\gamma(\boldsymbol{\theta}) \propto p(\boldsymbol{\theta})^\gamma \pi(\boldsymbol{\theta})^{1-\gamma}. \tag{4.5}$$

It is easily seen that the starting distribution $\pi(\boldsymbol{\theta})$ is recovered for $\gamma = 0$ and the target distribution $p(\boldsymbol{\theta})$ is recovered for $\gamma = 1$. Starting at $\gamma = 0$, we draw samples directly from $\pi(\boldsymbol{\theta})$ and set the weights $W^{(n)} = 1/N$; the goal being to gradually update the weights and the particle locations, in order to approximate the target distribution. The intermediate distributions p_γ for $\gamma \in (0, 1)$ provide a smooth transition path where importance sampling can be efficiently applied. In the example problems, $\pi(\boldsymbol{\theta})$ is chosen as the uniform distribution $\pi(\boldsymbol{\theta}) \propto 1$ in an Euclidean setting, and $p(\boldsymbol{\theta})$ is the uniform distribution (4.3) in the Riemannian setting described previously.

As the number of intermediate distributions is increased, we can expect an increased accuracy since the transition becomes smoother, but the computational cost also increases as more evaluations of $p(\boldsymbol{\theta})$ would be needed. On the other hand, too few intermediate distributions can adversely affect the overall accuracy of the particulate approximation. To that end we propose an adaptive SMC scheme as previously used in uncertainty quantification, stochastic design and system identification contexts [57, 58, 104], which determines automatically the number of needed intermediate distributions based on the *effective sample size*. The effective sample size is defined as $\mathrm{ESS} = 1/\sum_{n=1}^{N}(W^{(n)})^2 \in [1, N]$ and provides a measure of the variance of the importance weights [63]. Consequently, we define an acceptable reduction of the ESS from one distribution $\pi_{\gamma_{s-1}}$ to the next one π_{γ_s}, such that $\mathrm{ESS}(\gamma_s) \geqslant \zeta\,\mathrm{ESS}(\gamma_{s-1})$, with $\zeta < 1$, and then π_{γ_s} can be prescribed accordingly. Resampling the particles avoids a potential degeneracy of the algorithm when the ESS falls to low values; the extreme case being all the mass concentrated on a single particle of unit normalized weight, leading to $\mathrm{ESS} = 1$. Such a situation is prevented by resampling the particles with a probability corresponding to their weights, when some criterion is reached: usually when the effective sample size reaches $N/2$, half the number of particles. Then the importance weights are reset to $1/N$ to get $\mathrm{ESS} = N$. The details of the algorithm, as implemented in this work, appear in Algorithm 5.

It should be noted that unlike MCMC schemes, the particle perturbations in the rejuvenating step do not require the p_{γ_s}-invariant Metropolis-Hastings kernel $\mathcal{K}_s(\cdot,\cdot)$ to be ergodic [28], thus allowing adaptive change of its parameters to achieve better mixing rates. In this work, a classical random-walk Metropolis-Hastings kernel is used, although we mention that using the newly set geometric structure of the parameter space, as in [44], is a very interesting alternative.

Finally, we point out that the proposed adaptive SMC algorithm is embarrassingly par-

Algorithm 5: Adaptive Sequential Monte Carlo sampler

Initialization: Set $s = 1$, $\gamma_s = 0$, for all $n \in \{1, \dots, N\}$ draw $\boldsymbol{\theta}_1^{(n)} \sim \pi(\boldsymbol{\theta})$ and set all weights $\{W_1^{(n)}\}_{n=1..N}$ to $1/N$.

while $\gamma_s \neq 1$ **do**

Set $s \leftarrow s + 1$

Weights updating

Compute the unnormalized weights for each particle $n \in \{1, \dots, N\}$

$$w_s^{(n)}(\gamma_s) = W_{s-1}^{(n)} \frac{p_{\gamma_s}\left(\boldsymbol{\theta}_{s-1}^{(n)}\right)}{p_{\gamma_{s-1}}\left(\boldsymbol{\theta}_{s-1}^{(n)}\right)} = W_{s-1}^{(n)} \left(\frac{p\left(\boldsymbol{\theta}_{s-1}^{(n)}\right)}{\pi\left(\boldsymbol{\theta}_{s-1}^{(n)}\right)}\right)^{\gamma_s - \gamma_{s-1}}$$

where γ_s is found such that $\mathrm{ESS}(\gamma_s) = \zeta\, \mathrm{ESS}(\gamma_{s-1})$ ($\zeta = 0.95$ in the examples). Then normalize the weights to obtain $\{W_s^{(n)}\}_{n=1..N}$.

Resampling

Compute the effective sample size $\mathrm{ESS}(\gamma_s) = 1/\sum_{n=1}^{N} (W_s^{(n)})^2$ and resample if $\mathrm{ESS}(\gamma_s) < N/2$.

Rejuvenating

Perturbate the particles with a Metropolis-Hastings kernel $\mathcal{K}_s(\cdot, \cdot)$ of invariant distribution $p_{\gamma_s}(\boldsymbol{\theta})$, for $n \in \{1, \dots, N\}$ draw $\boldsymbol{\theta}_s^{(n)} \sim \mathcal{K}_s(\boldsymbol{\theta}_{s-1}^{(n)}, \cdot)$.

end while

allellizable, as the analysis for each particle can be performed independently during the rejuvenation step. An additional advantage of the proposed approach is that the algorithm outputs an empirical distribution approximating the target distribution, thus enabling its direct use as a starting distribution if a potential subsequent refinement of the metric tensor field approximation is needed.

4.3.3 Discussion of the sampling scheme

The interpolation scheme is integrated into the adaptive Sequential Monte Carlo scheme described previously, within the rejuvenation step and the associated Metropolis-Hastings kernel. The initial parameter value $\boldsymbol{\theta}_0$ is associated with the current reference metric tensor $\mathbf{I}_0$, and with a proposal move to $\boldsymbol{\theta}_*$ an interpolated metric tensor $\mathbf{I}_*$ is generated in order to compute accurately the quantity $\sqrt{\det \mathbf{I}(\boldsymbol{\theta})}$ in the accept-reject ratio. If the move to $\boldsymbol{\theta}_*$ is accepted, then the current reference metric tensor is set to $\mathbf{I}_*$. As such, no calls to the forward model is needed during the sampling process, enabling very fast computations once the precomputed population of Fisher information matrices $\{\mathbf{I}_1, \ldots, \mathbf{I}_K\}$ is generated. Indeed, the interpolation scheme makes only use of operations on $p \times p$ matrices, recalling that p is the size of the parameter vector $\boldsymbol{\theta}$.

Once the sampling process is completed, samples can readily be obtained from the particulate approximation (4.4) of the probability distribution $p(\boldsymbol{\theta}) \propto \sqrt{\det \mathbf{I}(\boldsymbol{\theta})}$. There is however an issue: samples from the uniform distribution on a Riemannian manifold can suffer from the same deficiencies as samples from the uniform distribution in an Euclidean space, the latter being significantly outperformed by low-discrepancy sequences. We advocate here the use of a clustering algorithm to alleviate this issue. From the weighted particles produced by the sampling process, we perform a resampling to get N independent samples of the uniform distribution on the parameter space, interpreted as a Riemannian manifold. A clustering algorithm such as *k-means* is then executed, using m as the number of clusters. In the following examples, we used the MATLAB function `kmeans` from the Statistics toolbox. The m centroids of the resulting tessellation are then the appropriate samples to be used to generate the snapshots.

4.4 Numerical examples

In this section, we will assess the performance of the proposed scheme in the context of reduced-order modeling. We propose here to use projection based reduced order models (ROMs), represented by their $n \times r$ orthogonal projection matrix $\mathbf{R}$, where n is the dimension of the full-order system and r is the dimension of the approximation subspace. Choices of reduced order modeling techniques include the *reduced basis method* [71, 79, 80] and *proper orthogonal decomposition* (POD, [50, 102] and references therein) or *principal components analysis* (PCA, [76, 51]). Such a projection matrix $\mathbf{R}$ is typically built using selected response snapshots from the full order model, corresponding to different values of the parameters $\boldsymbol{\theta}$. The present work adopts a POD approach to build those projection matrices. We consider a collection of m different responses from the full order model as an ensemble of snapshots; where each snapshot corresponds to a n-dimensional output vector, resulting in a $n \times m$ snapshot matrix $\mathbf{U}$:

$$\mathbf{U} = \begin{bmatrix} U_{11} & U_{12} & \cdots & U_{1m} \\ \vdots & \vdots & \ddots & \vdots \\ U_{n1} & U_{n2} & \cdots & U_{nm} \end{bmatrix}.$$

The POD modes are the eigenvectors of the covariance matrix $\mathbf{C} = \mathbf{U}\mathbf{U}^T$, or equivalently the left singular vectors of the snapshot matrix $\mathbf{U}$. The $n \times r$ orthogonal projection matrix $\mathbf{R}$ is then obtained from a subset of those modes corresponding to the r largest eigenvalues or singular values. Next, we describe the evaluation procedure of the proposed scheme, which we term *information geometric sampling* (IGS). We compare IGS and latin hypercube sampling (LHS) using a reference $n \times r$ projection matrix $\mathbf{R}_{\text{ref}}$, computed from a very large number of snapshots covering the entire parameter space and thus viewed as sort of ground truth. As such, the r columns of $\mathbf{R}_{\text{ref}}$ approximate closely the r modes incorporating the maximum information about the operator $\boldsymbol{\theta} \mapsto \boldsymbol{h}(\boldsymbol{u}(\boldsymbol{\theta}))$. For various numbers of snapshots

m, we will use the following error metric computed using the Frobenius matrix norm:

$$\mathcal{E}_{\text{IGS/LHS}} = \left\| \mathbf{R}_{\text{IGS/LHS}} - \mathbf{R}_{\text{ref}} \mathbf{R}_{\text{ref}}^T \mathbf{R}_{\text{IGS/LHS}} \right\|_{\text{fro}} \tag{4.6}$$

where $\mathbf{R}_{\text{IGS/LHS}}$ is the $n \times r$ projection matrix whose columns are the r dominant POD modes computed from m snapshots corresponding to a set of m parameters sampled using IGS or LHS. The error metric $\mathcal{E}_{\text{IGS/LHS}}$ can be interpreted as the magnitude of the orthogonal projection of the column vectors of $\mathbf{R}_{\text{IGS/LHS}}$ onto the orthogonal complement in $\mathbb{R}^n$ of the subspace spanned by the columns of $\mathbf{R}_{\text{ref}}$. As such, $\mathcal{E}_{\text{IGS/LHS}}$ provides a distance between the subspaces corresponding to the projection matrices $\mathbf{R}_{\text{IGS/LHS}}$ and $\mathbf{R}_{\text{ref}}$: the more similar they are, the smaller becomes $\mathcal{E}_{\text{IGS/LHS}}$. Greedy sampling methods such as [46, 17], using derivatives during the snapshot collection process, are subject to greater computational expenses; thus they are not included in this comparison. In the following, we apply this assessment procedure to two illustrative problems.

4.4.1 Heat source localization

The first numerical application consists in a simple heat equation problem on the square $\Omega = [-1, 1]^2$, parameterized as follows:

$$\begin{cases} \nabla \cdot (k(\boldsymbol{x}) \nabla u) + f(\boldsymbol{x}) = 0 & \text{in } \Omega \subset \mathbb{R}^2 \\ u(\boldsymbol{x}) = 0 & \text{on } \partial\Omega \subset \mathbb{R}^2 \end{cases} \tag{4.7}$$

where the conductivity and source terms are given by:

$$k(\boldsymbol{x}) = \exp\left(2x_1 + 2x_2^2\right) \quad \text{and} \quad f(\boldsymbol{x}) = A \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}_0\|^2}{2\tau^2}\right). \tag{4.8}$$

The parameter of interest is the source position $\boldsymbol{x}_0 = (x_{01}, x_{02})$, with $\boldsymbol{x}_0 \in [-1, 1]^2$. A profile of the conductivity term is given in Figure 4.2. Intuitively, we can conjecture that the most

"important" regions for the source position $\boldsymbol{x}_0$ will be the regions of lower conductivity, as the resulting peak in temperature would be more sharply peaked than in regions of lower conductivity.

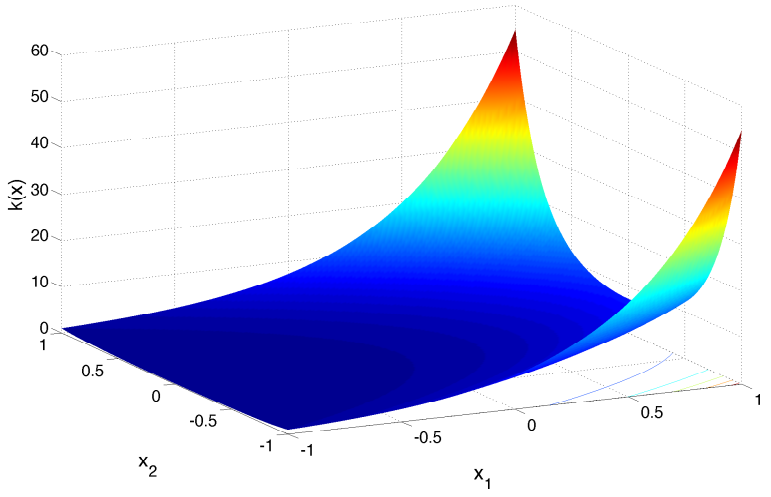


Figure 4.2: Conductivity profile from (4.8)

The problem (4.7) was discretized using the finite element method. We started by computing a population of ten metric tensors $\{\mathbf{I}_k\}_{k=1..10}$ in view of the interpolation procedure described in section 4.3.1. The corresponding parameters $\left\{\boldsymbol{x}_0^{(k)}\right\}_{k=1..10}$ were selected by latin hypercube sampling on the square $[-1,1]^2$. Those computations are the only additional expenses incurred by IGS compared to LHS. The adaptive Sequential Monte Carlo scheme described in section 4.3.2 was then run using 1,000 particles; corresponding samples from the resulting empirical distribution are plotted in Figure 4.3. The distribution of samples confirms our previous guess, in that regions of lower conductivity are emphasized, in which the temperature field is more sensitive to a change in the source position.

The parameters used to generate a number m of snapshots, taken in the range $m \in \{15, \ldots, 130\}$, were then obtained using a k-means clustering algorithm, as mentioned in

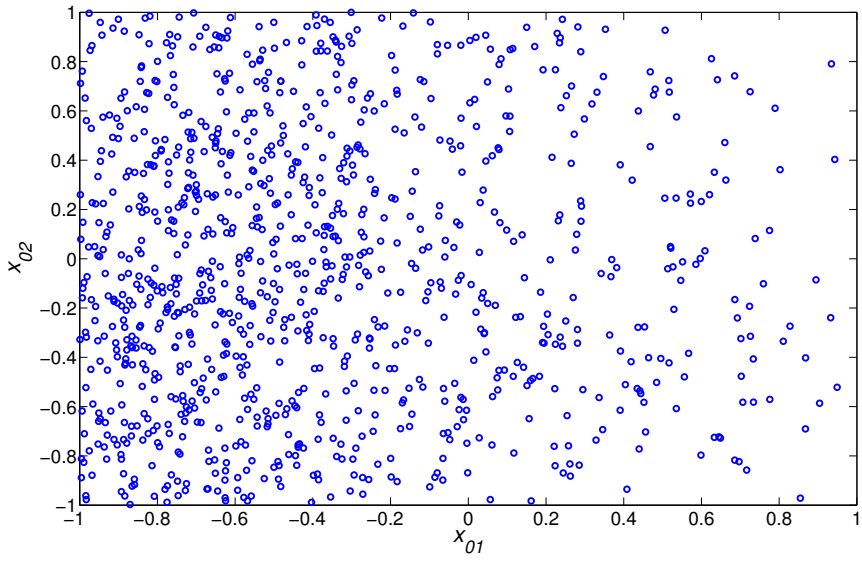
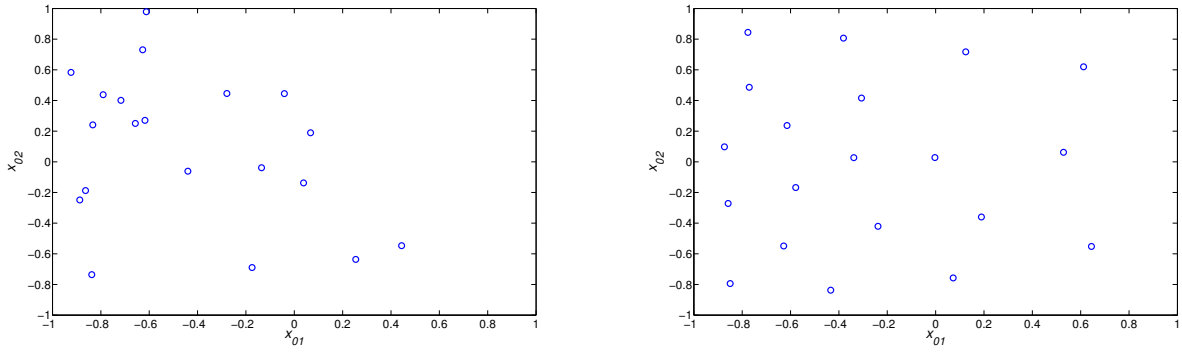


Figure 4.3: Samples from the particulate approximation of $p(\boldsymbol{\theta}) \propto \sqrt{\det \mathbf{I}(\boldsymbol{\theta})}$

section 4.3.3. The reduced basis matrix $\mathbf{R}_{\text{IGS}}$ is then formed by taking $r = 15$ POD modes following a singular value decomposition of the snapshot matrix, as to capture a sufficiently large proportion of the total variance. Figure 4.4 illustrates the value of clustering by displaying 20 samples taken directly from the empirical distribution via multinomial sampling, and the 20 centroids resulting from the k-means algorithm with $m = 20$: the samples from the latter scheme are less cluttered and represent better the whole parameter space as there are no "doubles" sampled for nearly the same parameter.

To compute the error metric $\mathcal{E}_{\text{IGS/LHS}}$ from (4.6), the reference reduced basis matrix $\mathbf{R}_{\text{ref}}$ is computed using 2,000 snapshots collected via latin hypercube sampling. $\mathbf{R}_{\text{ref}}$ is assumed to encapsulate as much information from those snapshots as can be in $r = 15$ modes. The error metric $\mathcal{E}_{\text{IGS}}$ is plotted in Figure 4.5 for the aforementioned range of number of snapshots, $m \in \{15, \ldots, 130\}$. We compare it with the error metric $\mathcal{E}_{\text{LHS}}$ for which the snapshots were computed from parameters collected using latin hypercube sampling, leading to a reduced



(a) 20 multinomial samples

(b) 20 cluster centroids

Figure 4.4: Comparison emphasizing the benefits of clustering the SMC output samples

basis matrix $\mathbf{R}_{\text{LHS}}$. As such sampling schemes naturally have inherent randomness, the average $\mathcal{E}_{\text{IGS/LHS}}$ over 25 runs is actually plotted, alongside their respective one standard deviation intervals. The subspace generated using IGS is then "closer" to the reference subspace spanned by the columns of $\mathbf{R}_{\text{ref}}$ than the subspace generated using LHS; thus enabling a better fidelity in reduced order modeling applications where a fixed computational effort is required. As the number of snapshots increases, it is expected that the IGS (red) and LHS (blue) errors become closer; with possibly the LHS error becoming smaller than the IGS error for a very large number of snapshots, since $\mathbf{R}_{\text{ref}}$ derives from latin hypercube sampling with 2,000 snapshots.

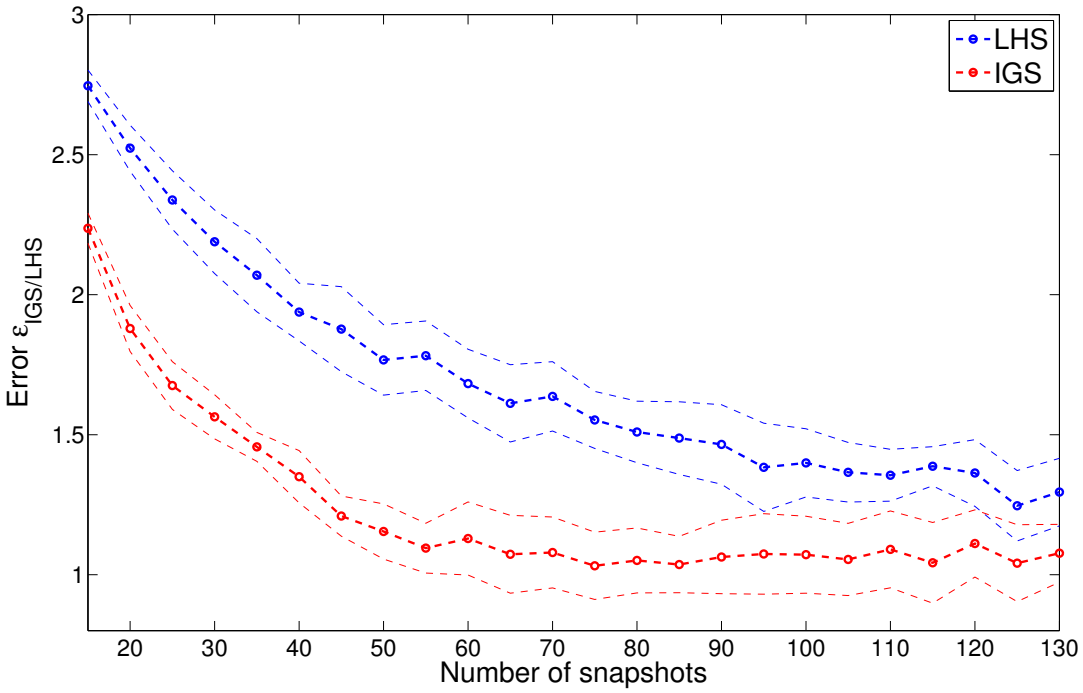


Figure 4.5: Average error metric $\mathcal{E}_{\text{IGS/LHS}}$ (4.6) against the number m of snapshots, with one standard deviation interval

4.4.2 Heat transfer in a bar

In this example, we consider once again the heat equation, but this time on a bar $\Omega = [-1, 1] \times [-0.25, 0.25]$. The problem and its boundary conditions, pictured in Figure 4.6, are parameterized as follows:

$$\begin{cases} \nabla \cdot (k(\boldsymbol{x}) \nabla u) = 0 & \text{in } \Omega \\ -k(\boldsymbol{x}) \frac{\partial u}{\partial n} = 0 & \text{on } \Sigma_0 \\ -k(\boldsymbol{x}) \frac{\partial u}{\partial n} = 5 & \text{on } \Sigma_q \\ u(\boldsymbol{x}) = 0 & \text{on } \Sigma_u \end{cases} \tag{4.9}$$

where the conductivity term is given by:

$$k(\boldsymbol{x}) = \exp\left(\sum_{i_1=1}^{5} c_{i_1}^{(1)} L_{i_1}(x_1) + \sum_{i_2=1}^{5} c_{i_2}^{(2)} L_{i_2}(4x_2)\right). \tag{4.10}$$

In the expression of the conductivity term (4.10), the $\{L_{i_{1,2}}\}_{i=1..5}$ are Legendre polynomials of order $i_{1,2}$; and their coefficients $\{c_{i_{1,2}}^{(1,2)}\}_{i=1..5} \in [-2,2]^{10}$ are the parameters of interest. Our problem is then to find the set of coefficients that best represents the whole range of conductivity fields which can be expressed as in (4.10).

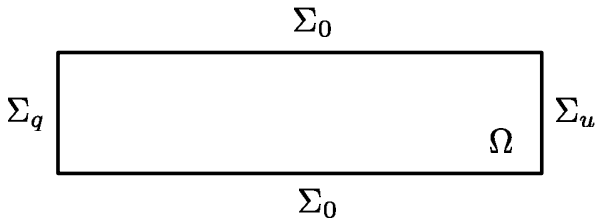


Figure 4.6: Configuration of the problem (4.9)

We discretized the problem (4.9) using the finite element method, and as in the previous example we first computed a population of 25 metric tensors $\{\mathbf{I}_k\}_{k=1..25}$ to approximate the metric field as in section 4.3.1; the corresponding parameters were selected by latin hypercube sampling on $[-2,2]^{10}$. The adaptive Sequential Monte Carlo sampling scheme described in section 4.3.2 was then run with 2,000 particles. The parameters of interest were then taken as the centroids resulting from a k-means clustering procedure (see section 4.3.3), with a number of clusters in the range $m \in \{15, \ldots, 130\}$. The reduced basis matrix $\mathbf{R}_{\text{IGS}}$ was then built using $r = 8$ POD modes with snapshots computed from the aforementioned parameters, once again as to capture a large enough proportion of the total variance. The matrix $\mathbf{R}_{\text{LHS}}$ used in the comparison was obtained similarly to the previous example, with latin hypercube sampling for the same numbers of snapshots $m \in \{15, \ldots, 130\}$. Finally, latin hypercube sampling was also used to collect 90,000 snapshots to build the reference

reduced basis matrix $\mathbf{R}_{\text{ref}}$ with $r = 8$ POD modes, to which $\mathbf{R}_{\text{IGS}}$ and $\mathbf{R}_{\text{LHS}}$ will be compared according to (4.6), using the error metric $\mathcal{E}_{\text{IGS/LHS}}$. As in the previous example, the average of $\mathcal{E}_{\text{IGS/LHS}}$ over 25 runs is plotted in Figure 4.7, along with their respective one standard deviation interval. Once again, and for a higher dimension of the parameter space, we can see that the use of IGS (red curve) brings substantial accuracy improvements over LHS (blue curve). The IGS error is consistently lower than the LHS error, and stabilizes at a low level for a much smaller number of snapshots; thus resulting in enhanced accuracy at a given computational effort.

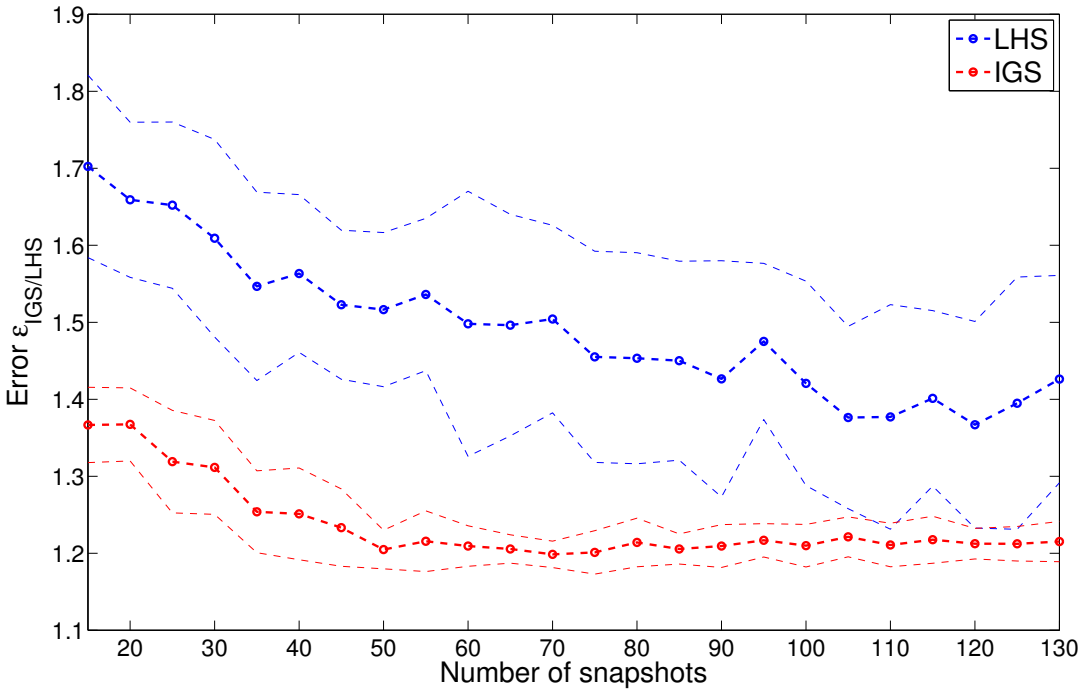


Figure 4.7: Average error metric $\mathcal{E}_{\text{IGS/LHS}}$ (4.6) against the number m of snapshots, with one standard deviation interval

4.5 Conclusions

The present chapter has described a framework for selecting suitable parameters to generate snapshots in view of reduced order modeling applications. The parameter space is interpreted as a Riemannian manifold equipped with a metric emphasizing its important regions. Sampling is performed through an efficient adaptive Sequential Monte Carlo scheme, in which the metric tensor field is approximated via an interpolation procedure; hence enabling fast computations without any call to the forward solver during the sampling process. The resulting parameters, obtained following a clustering procedure from the sampling output, are then used to generate snapshots. The example problems treated in this chapter demonstrate the promise of the proposed approach in displaying significant accuracy improvements while calling for low computational expenses controlled by the analyst.

CHAPTER 5

CONCLUSION AND FINAL THOUGHTS

This thesis was concerned with computational methods for uncertainty quantification, inverse problems and reduced order modeling. The goal of these methods is to accurately assess the consequences of uncertainties in the design and analysis of engineering systems, as well as to enable accurate and efficient system identification and prognosis; thus leading to a better understanding of the condition and and future performance of such systems.

5.1 Contributions

The work exposed in this thesis comprises three contributions to the aforementioned fields, together with illustrative and engineering-oriented numerical applications.

In chapter 2, a framework to optimize an engineering system under large uncertainties was described. The optimization problem was recast as a sampling problem, and the use of an advanced sampling scheme, associated with a hierarchical approach using approximate models, enabled an efficient identification of design values; along with corresponding sensitivity and robustness information. We applied successfully the devised approach to problems of stochastic design and control in the context of random heterogeneous materials, where uncertainties arise from the stochastic variability of their properties.

Chapter 3 presents a computationally efficient probabilistic framework that enables the identification of model parameters from noisy measurements of the response. We adopted a Bayesian approach, along with an advanced Sequential Monte Carlo sampling scheme and a reduced order model interpolation procedure based on differential geometric ideas. This enabled faster computations during the posterior sampling process, while maintaining

a high accuracy. The capabilities of the proposed framework were illustrated in the context of transient PDE-based models, in which the parameters correspond to physical properties.

In chapter 4, a novel methodology to sample effectively relevant points in a parameter space is devised. Of interest was the problem of collecting snapshots for reduced order modeling applications; namely how to select the parameters that represent best the system of interest in the parameter space. Using principles from information geometry, the latter was interpreted as a Riemannian manifold, equipped with a sensitivity related metric emphasizing the most informative regions. Comparative applications exemplify the effectiveness of the suggested approach for reduced order modeling. It is believed that the presented methodology can be extended to the broader field of statistical design of experiments.

A software library developed during the course of the research effort leading to this thesis is introduced in appendix A, and the corresponding documentation is given in appendix B, along with an illustrative example demonstrating the use of the library.

5.2 Final thoughts

This thesis has developed innovative computational tools for uncertainty quantification, inverse problems and reduced order modeling. Those fields emerged as new understanding concerning the behavior of engineering systems of ever-increasing complexity was needed. However, many questions remains without answer, and much progress remains to be done. It is believed that the constant advancement of uncertainty quantification and analysis methods, as well as of system identification and prognosis techniques, is of raising awareness in the engineering community and shows great promise for the future.

APPENDIX A

SOFTWARE CONTRIBUTION

Due to the heavy computational nature of the work described in this thesis, a significant part of software has been written. In this appendix, we introduce `SMCLib`, a template, object-oriented and extensible MCMC and SMC library written in C++. `SMCLib` has been developed during the course of the research work done in view of this thesis, in order to support the probabilistic methods described in the previous chapters. We will only discuss the software library in this appendix, as the mechanisms underlying Markov chain Monte Carlo methods, Sequential Monte Carlo methods and the adaptive Sequential Monte Carlo method has been described in the previous chapters and their references, such as [68, 48, 8, 83, 45, 65, 63, 31, 23, 28, 29, 57, 58, 104].

The goal was to establish a flexible and extensible library in order to expedite the development of a specific sampler for a particular application, such as the ones encountered in the previous chapters of this thesis. Similar efforts have been made in the statistical community, *e.g.* [52], however our approach differ in that we keep in mind the various requirements induced by the engineering applications described in this thesis. At this time of writing, implemented functionality includes MCMC, SMC and adaptive SMC samplers, Metropolis-Hastings and Metropolis-within-Gibbs kernels, as well as normal and multivariate normal probability distributions utilized as transition distributions.

The design of this library has been greatly inspired by software engineering design patterns such as the ones described in [69]: generic programming and object-oriented design have been used in order to increase flexibility and decrease the development time of a specific sampler. The use of templates enables to easily develop samplers for both univariate and multivariate probability distributions, while allowing for user-defined structure in the multi-

variate case. Object-oriented design accelerates the development of a particular application as specific samplers, kernels and probability distributions can be directly derived from library headers by using inheritance; with polymorphism enabling those derived objects to share the same global interface with the library. As we will see in section B.2, minimal coding work is required as much of the general functionality is included in the library.

The library makes heavy use of headers provided by the C++ Standard Library and the GNU Scientific Library. At the time of writing, `SMCLib` comprises the following elements:

- General and "high-level" template classes for MCMC and SMC samplers.
- General template classes representing elements composing the aforementioned samplers, such as MCMC kernels, particles and sets of particles.
- Abstract template classes representing probability distributions and sequences of probability distributions. Derived classes for univariate and multivariate normal distributions are included.

The documentation for those classes is included in section B.1. As always, there is room for improvements; specifically concerning ease of use and functionality. A major one would be to parallelize the SMC part of the library, using for instance MPI/OpenMP, so that the analysis for each particle is made on distinct processors. Potential extensions include supplementary probability distributions, as well as additional MCMC kernels such as Gibbs samplers [40], Hybrid Monte Carlo [35], the Metropolis-adjusted Langevin algorithm [13], etc.

APPENDIX B

DOCUMENTATION OF SMCLIB

B.1 Generated documentation

The following pages comprise the documentation, as generated from the source code by Doxygen 1.8.2 (`www.doxygen.org`), of the `SMCLib` software library described previously in Appendix A. It includes documentation for all member functions and data members for each class; and should prove itself as a useful reference for anyone wishing to use, modify or extend the `SMCLib` library. A simple example program is described in the next section, illustrating the use of the `SMCLib` library to sample from a univariate probability distribution.

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

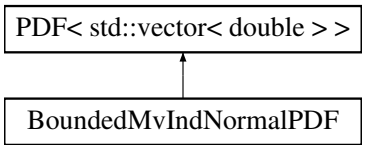
Class Documentation

3.1 BoundedMvIndNormalPDF Class Reference

Inherited class representing a bounded multivariate Gaussian distribution.

`#include <BoundedMvIndNormalPDF.h>`

Inheritance diagram for BoundedMvIndNormalPDF:



Public Member Functions

- **BoundedMvIndNormalPDF** (long d)

  Constructor of the class.
- **BoundedMvIndNormalPDF** (long d, const std::vector< double > &mean, const std::vector< double > &std_dev, const std::vector< double > &bc1, const std::vector< double > &bc2)

  Alternative constructor.
- virtual ~**BoundedMvIndNormalPDF** ()

  Destructor of the class.
- void **setDimension** (long dim)

  Set the dimension of the state space.
- long **getDimension** () const

  Get the dimension of the state space.
- void **setMeanVector** (const std::vector< double > &mean)

  Set the mean vector of the distribution.
- std::vector< double > **getMeanVector** () const

  Get the mean vector of the distribution.
- void **setStdDevVector** (const std::vector< double > &std_dev)

  Set the standard deviation vector of the distribution.
- std::vector< double > **getStdDevVector** () const

Get the standard deviation vector of the distribution.

- void **setInfBoundVector** (const std::vector< double > &bc1)

  Set the inferior boundary vector.
- std::vector< double > **getInfBoundVector** () const

  Get the inferior boundary vector.
- void **setSupBoundVector** (const std::vector< double > &bc2)

  Set the superior boundary vector.
- std::vector< double > **getSupBoundVector** () const

  Get the superior boundary vector.
- void **setMeanVectorComponent** (double mean_i, long i)

  Set one component of the mean vector of the distribution.
- double **getMeanVectorComponent** (long i) const

  Get one component of the mean vector of the distribution.
- void **setStdDevVectorComponent** (double std_dev_i, long i)

  Set one component of the standard deviation vector of the distribution.
- double **getStdDevVectorComponent** (long i) const

  Get one component of the standard deviation vector of the distribution.
- void **setInfBoundVectorComponent** (double bc1_i, long i)

  Set one component of the inferior boundary vector.
- double **getInfBoundVectorComponent** (long i) const

  Get one component of the inferior boundary vector.
- void **setSupBoundVectorComponent** (double bc2_i, long i)

  Set one component of the superior boundary vector.
- double **getSupBoundVectorComponent** (long i) const

  Get one component of the superior boundary vector.
- virtual double **evalPdf** (const std::vector< double > &X) const

  Evaluate the pdf at the vector X.
- virtual double **evalCdf** (const std::vector< double > &X) const

  Evaluate the cdf at the vector X.
- virtual std::vector< double > **evalInvCdf** (double) const

  Dummy overriding of the inverse cdf member function.
- virtual std::vector< double > **getSample** () const

  Get a sample vector from the distribution.
- virtual double **evalPdf** (const std::vector< double > &X, const std::vector< double > &Y) const

  Evaluate the kernel pdf from X to Y.
- virtual double **evalCdf** (const std::vector< double > &X, const std::vector< double > &Y) const

  Evaluate the kernel cdf from X to Y.
- virtual std::vector< double > **evalInvCdf** (double, const std::vector< double > &) const

  Dummy overriding of the inverse kernel cdf member function.
- virtual std::vector< double > **getSample** (const std::vector< double > &X) const

  Get a sample from the kernel distribution from X.
- virtual std::vector< double > **getSample** (const std::vector< double > &X, long i) const

  Get a sample from the kernel distribution by modifying only the ith component.
- double **evalMarginalPdf** (double x, long i) const

  Evaluate the ith marginal pdf of the distribution at x.
- double **evalMarginalCdf** (double x, long i) const

  Evaluate the ith marginal cdf of the distribution at x.

- double **evalMarginalInvCdf** (double p, long i) const

  Dummy overriding – Evaluate the inverse cdf of the ith marginal distribution at probability p.
- double **getMarginalSample** (long i) const

  Get a sample from the ith marginal distribution.
- double **evalMarginalPdf** (double x, double y, long i) const

  Evaluate the ith marginal kernel pdf from x to y.
- double **evalMarginalCdf** (double x, double y, long i) const

  Evaluate the ith marginal kernel cdf from x to y.
- double **evalMarginalInvCdf** (double p, double x, long i) const

  Dummy overriding – Evaluate the inverse kernel cdf from x of the ith marginal at probability p.
- double **getMarginalSample** (double x, long i) const

  Get a sample from the ith kernel marginal distribution from x.
- virtual void **setRandNumGen** (gsl_rng ∗r)

  Set the random number generator.

Protected Attributes

- **MvIndNormalPDF** ∗ **mvn**

  Underlying multivariate normal distribution.
- std::vector< double > **bound1**

  Inferior boundary of the samples.
- std::vector< double > **bound2**

  Superior boundary of the samples.

3.1.1 Detailed Description

Inherited class representing a bounded multivariate Gaussian distribution.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 BoundedMvIndNormalPDF::BoundedMvIndNormalPDF (long *d*)

Constructor of the class.

Default constructor, set the mean to 0 and the standard deviation to 1 for the boundaries [-1,1].

3.1.2.2 BoundedMvIndNormalPDF::BoundedMvIndNormalPDF (long *d*, const std::vector< double > & *mean*, const std::vector< double > & *std_dev*, const std::vector< double > & *bc1*, const std::vector< double > & *bc2*)

Alternative constructor.

Enable the user to define the mean, standard deviation and bounds.

3.1.2.3 BoundedMvIndNormalPDF::∼BoundedMvIndNormalPDF ( ) [virtual]

Destructor of the class.

Destructor - Delete the underlying multivariate normal.

3.1.3 Member Function Documentation

3.1.3.1 double BoundedMvIndNormalPDF::evalCdf (const std::vector< double > & *X*) const [virtual]

Evaluate the cdf at the vector X.

Return the cdf value for the bounded multivariate normal.

Implements **PDF< std::vector< double > >** (p. 123).

3.1.3.2 double BoundedMvIndNormalPDF::evalCdf (const std::vector< double > & *X*, const std::vector< double > & *Y*) const [virtual]

Evaluate the kernel cdf from X to Y.

Return the kernel cdf value for the bounded multivariate normal.

Implements **PDF< std::vector< double > >** (p. 123).

3.1.3.3 double BoundedMvIndNormalPDF::evalMarginalCdf (double *x*, long *i*) const

Evaluate the ith marginal cdf of the distribution at x.

Return the ith marginal cdf value using the underlying multivariate normal.

3.1.3.4 double BoundedMvIndNormalPDF::evalMarginalCdf (double *x*, double *y*, long *i*) const

Evaluate the ith marginal kernel cdf from x to y.

Return the ith marginal kernel cdf value using the underlying multivariate normal.

3.1.3.5 double BoundedMvIndNormalPDF::evalMarginalPdf (double *x*, long *i*) const

Evaluate the ith marginal pdf of the distribution at x.

Return the ith marginal pdf value using the underlying multivariate normal.

3.1.3.6 double BoundedMvIndNormalPDF::evalMarginalPdf (double *x*, double *y*, long *i*) const

Evaluate the ith marginal kernel pdf from x to y.

Return the ith marginal kernel pdf value using the underlying multivariate normal.

3.1.3.7 double BoundedMvIndNormalPDF::evalPdf (const std::vector< double > & *X*) const [virtual]

Evaluate the pdf at the vector X.

Return the pdf value for the bounded multivariate normal.

Implements **PDF< std::vector< double > >** (p. 123).

3.1.3.8 double BoundedMvIndNormalPDF::evalPdf (const std::vector< double > & *X*, const std::vector< double > & *Y*) const [virtual]

Evaluate the kernel pdf from X to Y.

Return the kernel pdf value for the bounded multivariate normal.

Implements **PDF< std::vector< double > >** (p. 123).

3.1.3.9 long BoundedMvIndNormalPDF::getDimension ( ) const

Get the dimension of the state space.

Get the dimension from the underlying multivariate normal.

3.1.3.10 std::vector< double > BoundedMvIndNormalPDF::getInfBoundVector ( ) const

Get the inferior boundary vector.

Return a copy of the inferior boundary vector.

3.1.3.11 double BoundedMvIndNormalPDF::getInfBoundVectorComponent (long *i*) const

Get one component of the inferior boundary vector.

Access the inferior boundary vector to return one element.

3.1.3.12 double BoundedMvIndNormalPDF::getMarginalSample (long *i*) const

Get a sample from the ith marginal distribution.

Return a sample from the ith marginal distribution using the underlying multivariate normal.

3.1.3.13 double BoundedMvIndNormalPDF::getMarginalSample (double *x*, long *i*) const

Get a sample from the ith kernel marginal distribution from x.

Return a sample from the ith marginal kernel distribution using the underlying multivariate normal.

3.1.3.14 std::vector< double > BoundedMvIndNormalPDF::getMeanVector ( ) const

Get the mean vector of the distribution.

Get the mean vector from the underlying multivariate normal.

3.1.3.15 std::vector< double > BoundedMvIndNormalPDF::getSample ( ) const [virtual]

Get a sample vector from the distribution.

Get a sample using the underlying multivariate normal.

Implements **PDF< std::vector< double > >** (p. 123).

3.1.3.16 std::vector< double > BoundedMvIndNormalPDF::getSample (const std::vector< double > & *X*) const [virtual]

Get a sample from the kernel distribution from X.

Get a sample from the kernel distribution using the underlying multivariate normal.

Implements **PDF< std::vector< double > >** (p. 123).

3.1.3.17 std::vector< double > BoundedMvIndNormalPDF::getSample (const std::vector< double > & *X*, long *i*) const [virtual]

Get a sample from the kernel distribution by modifying only the ith component.

Get a sample from the kernel distribution by updating only the ith component using the underlying multivariate normal.

Implements **PDF< std::vector< double > >** (p. 123).

3.1.3.18 std::vector< double > BoundedMvIndNormalPDF::getStdDevVector ( ) const

Get the standard deviation vector of the distribution.

Get the standard deviation vector from the underlying multivariate normal.

3.1.3.19 double BoundedMvIndNormalPDF::getStdDevVectorComponent (long *i*) const

Get one component of the standard deviation vector of the distribution.

Get one component from the standard deviation vector of the underlying multivariate normal.

3.1.3.20 std::vector< double > BoundedMvIndNormalPDF::getSupBoundVector ( ) const

Get the superior boundary vector.

Return a copy of the superior boundary vector.

3.1.3.21 double BoundedMvIndNormalPDF::getSupBoundVectorComponent (long *i*) const

Get one component of the superior boundary vector.

Access the superior boundary vector to return one element.

3.1.3.22 void BoundedMvIndNormalPDF::setDimension (long *dim*)

Set the dimension of the state space.

Set the dimension in the underlying multivariate normal.

3.1.3.23 void BoundedMvIndNormalPDF::setInfBoundVector (const std::vector< double > & *bc1*)

Set the inferior boundary vector.

Copy the argument vector into the inferior boundary vector.

3.1.3.24 void BoundedMvIndNormalPDF::setInfBoundVectorComponent (double *bc1_i,* long *i*)

Set one component of the inferior boundary vector.

Access the inferior boundary vector to change one element.

3.1.3.25 void BoundedMvIndNormalPDF::setMeanVector (const std::vector< double > & *mean*)

Set the mean vector of the distribution.

Copy the argument vector in the mean vector of the underlying multivariate normal.

3.1.3.26 void BoundedMvIndNormalPDF::setMeanVectorComponent (double *mean_i,* long *i*)

Set one component of the mean vector of the distribution.

Change one element in the mean vector of the underlying multivariate normal.

3.1.3.27 void BoundedMvIndNormalPDF::setStdDevVector (const std::vector< double > & *std_dev*)

Set the standard deviation vector of the distribution.

Copy the argument vector in the standard deviation vector of the underlying multivariate normal.

3.1.3.28 void BoundedMvIndNormalPDF::setStdDevVectorComponent (double *std_dev_i,* long *i*)

Set one component of the standard deviation vector of the distribution.

Change one element in the standard deviation vector of the underlying multivariate normal.

3.1.3.29 void BoundedMvIndNormalPDF::setSupBoundVector (const std::vector< double > & *bc2*)

Set the superior boundary vector.

Copy the argument vector into the superior boundary vector.

3.1.3.30 void BoundedMvIndNormalPDF::setSupBoundVectorComponent (double *bc2_i,* long *i*)

Set one component of the superior boundary vector.

Access the superior boundary vector to change one element.

The documentation for this class was generated from the following files:

- BoundedMvIndNormalPDF.h
- BoundedMvIndNormalPDF.cpp

3.2 MCMCSampler< T > Class Template Reference

Template base class for a McMC sampler.

`#include <MCMCSampler.h>`

Public Member Functions

- **MCMCSampler** ()
  Constructor of the class.
- **MCMCSampler** (long numiter, long burnin)
  Alternative constructor.
- virtual ~**MCMCSampler** ()
  Destructor of the class.
- void **setInitialState** (const T &init)
  Set the initial state.
- void **setNumIterations** (long numiter)
  Set the number of iterations.
- long **getNumIterations** () const
  Get the number of iterations.
- void **setBurnInIterations** (long burnin)
  Set the number of burn-in iterations.
- long **getBurnInIterations** () const
  Get the number of burn-in iterations.
- void **setCurrentIterNum** (long iter)
  Set the current iteration number.
- void **addCurrentIterNum** ()
  Add one iteration to the current iteration number.
- long **getCurrentIterNum** () const
  Get the current iteration number.
- void **setNumAcceptedMoves** (long numacc)
  Set the number of accepted McMC moves.
- void **addToNumAcceptedMoves** (long increment)
  Add some number to the number of accepted moves.
- long **getNumAcceptedMoves** () const
  Get the number of accepted moves.
- double **getAcceptanceRatio** () const
  Get the current acceptance ratio.
- void **setMcmcUpdate** (**MCMCUpdate**< T > &mcmc)
  Set the McMC update to use.
- void **setNumMcmcMoves** (long nummoves)
  Set the number of McMC moves per iteration.
- long **getNumMcmcMoves** () const
  Get the number of McMC moves per iteration.
- void **setRandNumGen** (gsl_rng *r)
  Set the random number generator.
- void **setTargetPdf** (**PDF**< T > &tpdf)
  Set the pointer to the target pdf to be sampled.
- void **setTransPdf** (**PDF**< T > &tpdf)
  Set the pointer to the transition pdf to use.
- virtual void **readInputParameters** ()
  Read input parameters from a file.
- virtual void **runMcmcSampler** ()
  Run the McMC sampler, main member function.
- virtual void **outputReport** ()
  Output results on screen and in a file.

Protected Attributes

- std::vector< T > **samples**

  Vector containing the samples.
- T **initial_state**

  Initial state.
- long **num_iterations**

  Number of iterations.
- long **burn_in_iterations**

  Number of burn-in iterations.
- long **current_iter**

  Current iteration number.
- long **num_accepted_moves**

  Number of accepted McMC moves.
- **MCMCUpdate**< T > * **mcmc_update**

  Pointer to a McMC update object.
- long **num_mcmc_moves**

  Number of McMC moves per iteration.
- gsl_rng * **rand_gen**

  Pointer to an instance of a eandom number generator.
- **PDF**< T > * **target_pdf**

  *Pointer to a **PDF** (p. 123) object representing the target pdf.*
- **PDF**< T > * **trans_pdf**

  Pointer to a pdf object representing the transition pdf.

3.2.1 Detailed Description

template<typename T>class MCMCSampler< T >

Template base class for a McMC sampler.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 template<typename T > MCMCSampler< T >::MCMCSampler ( )

Constructor of the class.

Default constructor, set parameters to defined values.

3.2.2.2 template<typename T > MCMCSampler< T >::MCMCSampler (long *numiter,* long *burnin*)

Alternative constructor.

Set the number of iterations, burn-in time and number of moves to defined values.

3.2.3 Member Function Documentation

3.2.3.1 template<typename T > double MCMCSampler< T >::getAcceptanceRatio ( ) const

Get the current acceptance ratio.

Return the value obtained by the formula $n_{accepted}/n_{moves}n_{iter}$.

3.2.3.2 template<typename T > void MCMCSampler< T >::outputReport ( ) [virtual]

Output results on screen and in a file.

Output the problem data, to be overriden.

3.2.3.3 template<typename T > void MCMCSampler< T >::readInputParameters ( ) [virtual]

Read input parameters from a file.

Open an input file and read parameters.

3.2.3.4 template<typename T > void MCMCSampler< T >::runMcmcSampler ( ) [virtual]

Run the McMC sampler, main member function.

Main function of the class.

3.2.3.5 template<typename T > void MCMCSampler< T >::setNumMcmcMoves (long *nummoves*)

Set the number of McMC moves per iteration.

Modifies the number of McMC moves, also in the McMC update object.

3.2.3.6 template<typename T > void MCMCSampler< T >::setRandNumGen (gsl_rng * *r*)

Set the random number generator.

Set the random number generator in the sub-objects.

3.2.3.7 template<typename T > void MCMCSampler< T >::setTargetPdf (PDF< T > & *tpdf*)

Set the pointer to the target pdf to be sampled.

Set the target pdf, also modify the McMC update object.

3.2.3.8 template<typename T > void MCMCSampler< T >::setTransPdf (PDF< T > & *tpdf*)

Set the pointer to the transition pdf to use.

Set the transition pdf, also modify the McMC update object.

The documentation for this class was generated from the following file:

- MCMCSampler.h

3.3 MCMCUpdate< T > Class Template Reference

Template class for McMC updates.

`#include <MCMCUpdate.h>`

Public Member Functions

- **MCMCUpdate** ()
  Constructor of the class.
- **MCMCUpdate** (long ns)
  Alternative constructor.
- virtual ~**MCMCUpdate** ()
  Destructor of the class.
- void **setNsteps** (long n)
  Set the number of McMC moves.
- long **getNsteps** () const
  Get the number of McMC moves.
- void **setCurrentState** (const T &)
  Set the current state.
- T **getCurrentState** () const
  Get the current state.
- void **setProposalState** (const T &)
  Set the proposal state.
- T **getProposalState** () const
  Get the proposal state.
- void **setTargetPdfSequence** (**PDFSequence**< T > &)
  Set the pointer to the target sequence of probability distributions (SMC)
- void **setTargetPdf** (**PDF**< T > &)
  Set the pointer to the target probability distribution function.
- void **setTransPdf** (**PDF**< T > &)
  Set the pointer to the transition probability distribution function.
- void **setNumAccept** (long n)
  Set the number of accepted moves.
- long **getNumAccept** () const
  Get the number of accepted moves.
- void **addNumAccept** ()
  Increment the number of accepted moves.
- void **setRandNumGen** (gsl_rng *r)
  Set the random number generator.
- virtual void **performMcmcUpdate** ()
  Perform the McMC update procedure.
- virtual void **performMcmcUpdateInSmc** ()
  Perform the McMC update procedure (SMC)
- virtual void **performMcmcUpdateInSmc** (double &fun_value)
  Perform the McMC update procedure with an expensive function (SMC)

Protected Attributes

- long **n_steps**

  Number of McMC moves.
- long **num_accept**

  Number of accepted moves.
- T **current_state**

  Current state.
- T **proposal_state**

  Proposal state.
- gsl_rng * **rand_gen**

  Pointer to an instance of a random number generator.
- **PDFSequence**< T > * **target_pdf_sequence**

  *Pointer to a **PDFSequence** (p. 124) object representing the target pdf sequence (SMC)*
- **PDF**< T > * **target_pdf**

  *Pointer to a **PDF** (p. 123) object representing the target pdf.*
- **PDF**< T > * **trans_pdf**

  *Pointer to a **PDF** (p. 123) object representing the transition pdf.*

3.3.1 Detailed Description

template<typename T>class MCMCUpdate< T >

Template class for McMC updates.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 template<typename T > MCMCUpdate< T >::MCMCUpdate ( )

Constructor of the class.

Set the number of steps to a default value.

3.3.2.2 template<typename T > MCMCUpdate< T >::MCMCUpdate (long *ns*)

Alternative constructor.

Set the number of steps to a defined value.

3.3.3 Member Function Documentation

3.3.3.1 template<typename T > T MCMCUpdate< T >::getCurrentState ( ) const

Get the current state.

Return a copy of the current state.

3.3.3.2 template<typename T > T MCMCUpdate< T >::getProposalState ( ) const

Get the proposal state.

Return a copy of the proposal state.

3.3.3.3 template<typename T > void MCMCUpdate< T >::performMcmcUpdate ( ) [virtual]

Perform the McMC update procedure.

Classical Metropolis-Hastings algorithm.

Reimplemented in **MetWGibbsUpdate< T >** (p. 111).

3.3.3.4 template<typename T > void MCMCUpdate< T >::performMcmcUpdateInSmc ( ) [virtual]

Perform the McMC update procedure (SMC)

Classical Metropolis-Hastings algorithm in a SMC framework.

Reimplemented in **MetWGibbsUpdate< T >** (p. 111).

3.3.3.5 template<typename T > void MCMCUpdate< T >::performMcmcUpdateInSmc (double & *fun_value*) [virtual]

Perform the McMC update procedure with an expensive function (SMC)

Classical Metropolis-Hastings algorithm in a SMC framework with an expensive function, modifies the argument fun_value.

Reimplemented in **MetWGibbsUpdate< T >** (p. 111).

3.3.3.6 template<typename T> void MCMCUpdate< T >::setCurrentState (const T & *cstate*)

Set the current state.

Copy the argument into the current state.

3.3.3.7 template<typename T> void MCMCUpdate< T >::setProposalState (const T & *pstate*)

Set the proposal state.

Copy the argument vector into the proposal state.

3.3.3.8 template<typename T> void MCMCUpdate< T >::setTargetPdf (PDF< T > & *tpdf*)

Set the pointer to the target probability distribution function.

Target invariant probability distribution to be sampled.

3.3.3.9 template<typename T> void MCMCUpdate< T >::setTargetPdfSequence (PDFSequence< T > & *tpdfseq*)

Set the pointer to the target sequence of probability distributions (SMC)

Target invariant probability distribution to be sampled in a SMC framework.

3.3.3.10 template<typename T> void MCMCUpdate< T >::setTransPdf (PDF< T > & *tpdf*)

Set the pointer to the transition probability distribution function.

Transition probability distribution of the McMC update.

The documentation for this class was generated from the following file:

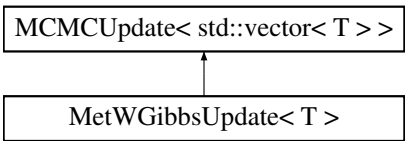
- MCMCUpdate.h

3.4 MetWGibbsUpdate< T > Class Template Reference

Template class for Metropolis-within-Gibbs updates.

`#include <MetWGibbsUpdate.h>`

Inheritance diagram for MetWGibbsUpdate< T >:



Public Member Functions

- **MetWGibbsUpdate** ()
  Constructor of the class.
- **MetWGibbsUpdate** (long ns, long ng)
  Alternative constructor.
- virtual ~**MetWGibbsUpdate** ()
  Destructor of the class.
- void **setNumAcceptGroup** (long n, long k)
  Set the number of accepted moves for one group of components.
- void **addNumAcceptGroup** (long k)
  Add one to the number of accepted moves for one group of components.
- long **getNumAcceptGroup** (long k) const
  Get the number of accepted moves for one group of components.
- void **setNumGroups** (long ng)
  Set the number of separate groups.
- long **getNumGroups** () const
  Get the number of separate groups.
- virtual void **performMcmcUpdate** ()
  Perform the update procedure.
- virtual void **performMcmcUpdateInSmc** ()
  Perform the update procedure (SMC)
- virtual void **performMcmcUpdateInSmc** (double &fun_value)
  Perform the update procedure with an expensive function (SMC)

Protected Attributes

- long **ngroups**

  Number of separate groups.
- std::vector< long > **num_accept_group**

  Number of accepted moves per group of components.

3.4.1 Detailed Description

template<typename T>class MetWGibbsUpdate< T >

Template class for Metropolis-within-Gibbs updates.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 template<typename T > MetWGibbsUpdate< T >::MetWGibbsUpdate ( )

Constructor of the class.

Set the number of steps to a default value.

3.4.2.2 template<typename T > MetWGibbsUpdate< T >::MetWGibbsUpdate (long *ns*, long *ng*)

Alternative constructor.

Set the number of steps to a defined value.

3.4.3 Member Function Documentation

3.4.3.1 template<typename T > void MetWGibbsUpdate< T >::addNumAcceptGroup (long *k*)

Add one to the number of accepted moves for one group of components.

Increment by one the number of accepted moves for group k.

3.4.3.2 template<typename T > long MetWGibbsUpdate< T >::getNumAcceptGroup (long *k*) const

Get the number of accepted moves for one group of components.

Return the number of accepted moves for group k.

3.4.3.3 template<typename T > long MetWGibbsUpdate< T >::getNumGroups ( ) const

Get the number of separate groups.

Get the number of groups to be updated separately.

3.4.3.4 template<typename T > void MetWGibbsUpdate< T >::performMcmcUpdate ( ) [virtual]

Perform the update procedure.

Metropolis-within-Gibbs algorithm.

Reimplemented from **MCMCUpdate**< **std::vector**< **T** > > (p. 108).

3.4.3.5 template<typename T > void MetWGibbsUpdate< T >::performMcmcUpdateInSmc ( ) [virtual]

Perform the update procedure (SMC)

Metropolis-within-Gibbs algorithm in a SMC framework.

Reimplemented from **MCMCUpdate**< **std::vector**< **T** > > (p. 108).

3.4.3.6 template<typename T > void MetWGibbsUpdate< T >::performMcmcUpdateInSmc (double & *fun_value*) [virtual]

Perform the update procedure with an expensive function (SMC)

Metropolis-within-Gibbs algorithm in a SMC framework with an expensive function.

Reimplemented from **MCMCUpdate**< **std::vector**< **T** > > (p. 108).

3.4.3.7 template<typename T > void MetWGibbsUpdate< T >::setNumAcceptGroup (long *n,* long *k*)

Set the number of accepted moves for one group of components.

Set the number of accepted moves for group k.

3.4.3.8 template<typename T > void MetWGibbsUpdate< T >::setNumGroups (long *ng*)

Set the number of separate groups.

Set the number of groups to be updated separately.

The documentation for this class was generated from the following file:

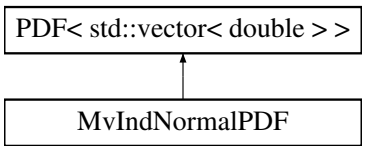
- MetWGibbsUpdate.h

3.5 MvIndNormalPDF Class Reference

Inherited class representing an multivariate Gaussian distribution.

`#include <MvIndNormalPDF.h>`

Inheritance diagram for MvIndNormalPDF:



Public Member Functions

- **MvIndNormalPDF** (long d)

  Constructor of the class.
- **MvIndNormalPDF** (long d, const std::vector< double > &mean, const std::vector< double > &std_dev)

  Alternative constructor.
- virtual ~**MvIndNormalPDF** ()

  Destructor of the class.
- void **setDimension** (long dim)

  Set the dimension of the state space.
- long **getDimension** () const

  Get the dimension of the state space.
- void **setMeanVector** (const std::vector< double > &mean)

  Set the mean vector of the normal distribution.
- std::vector< double > **getMeanVector** () const

  Get the mean vector of the normal distribution.
- void **setStdDevVector** (const std::vector< double > &std_dev)

  Set the standard deviation vector of the normal distribution.
- std::vector< double > **getStdDevVector** () const

  Get the standard deviation vector of the normal distribution.
- void **setMeanVectorComponent** (double mean_i, long i)

  Set one component of the mean vector of the normal distribution.
- double **getMeanVectorComponent** (long i) const

  Get one component of the mean vector of the normal distribution.
- void **setStdDevVectorComponent** (double std_dev_i, long i)

  Set one component of the standard deviation vector of the normal distribution.
- double **getStdDevVectorComponent** (long i) const

  Get one component of the standard deviation vector of the normal distribution.
- virtual double **evalPdf** (const std::vector< double > &X) const

  Evaluate the pdf at the vector X.
- virtual double **evalCdf** (const std::vector< double > &X) const

  Evaluate the cdf at the vector X.
- virtual std::vector< double > **evalInvCdf** (double) const

  Dummy overriding of the inverse cdf member function.
- virtual std::vector< double > **getSample** () const

  Get a sample vector from the distribution.
- virtual double **evalPdf** (const std::vector< double > &X, const std::vector< double > &Y) const

  Evaluate the kernel pdf from X to Y.
- virtual double **evalCdf** (const std::vector< double > &X, const std::vector< double > &Y) const

  Evaluate the kernel cdf from X to Y.
- virtual std::vector< double > **evalInvCdf** (double, const std::vector< double > &) const

  Dummy overriding of the inverse kernel cdf member function.
- virtual std::vector< double > **getSample** (const std::vector< double > &X) const

  Get a sample from the kernel distribution from X.
- virtual std::vector< double > **getSample** (const std::vector< double > &X, long i) const

  Get a sample from the kernel distribution by modifying only the ith component.
- double **evalMarginalPdf** (double x, long i) const

Evaluate the ith marginal pdf of the distribution at x.

- double **evalMarginalCdf** (double x, long i) const

  Evaluate the ith marginal cdf of the distribution at x.
- double **evalMarginalInvCdf** (double p, long i) const

  Evaluate the inverse cdf of the ith marginal distribution at probability p.
- double **getMarginalSample** (long i) const

  Get a sample from the ith marginal distribution.
- double **evalMarginalPdf** (double x, double y, long i) const

  Evaluate the ith marginal kernel pdf from x to y.
- double **evalMarginalCdf** (double x, double y, long i) const

  Evaluate the ith marginal kernel cdf from x to y.
- double **evalMarginalInvCdf** (double p, double x, long i) const

  Evaluate the inverse kernel cdf from x of the ith marginal at probability p.
- double **getMarginalSample** (double x, long i) const

  Get a sample from the ith kernel marginal distribution from x.

Protected Attributes

- long **dimension**

  Dimension of the state space.
- std::vector< double > **mu**

  Mean vector of the normal distribution.
- std::vector< double > **sigma**

  Standard deviation vector of the normal distribution.

3.5.1 Detailed Description

Inherited class representing an multivariate Gaussian distribution.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 MvIndNormalPDF::MvIndNormalPDF (long *d*)

Constructor of the class.

Default constructor, set the mean to 0 and the standard deviation to 1.

3.5.2.2 MvIndNormalPDF::MvIndNormalPDF (long *d*, const std::vector< double > & *mean*, const std::vector< double > & *std_dev*)

Alternative constructor.

Enable the user to define the mean and the standard deviation vectors.

3.5.3 Member Function Documentation

3.5.3.1 double MvIndNormalPDF::evalCdf (const std::vector< double > & *X*) const [virtual]

Evaluate the cdf at the vector X.

Return the value obtained by the formula $P(X) = \int_{-\infty}^{X} p(X')dX'$.

Implements **PDF< std::vector< double > >** (p. 123).

3.5.3.2 double MvIndNormalPDF::evalCdf (const std::vector< double > & *X*, const std::vector< double > & *Y*) const [virtual]

Evaluate the kernel cdf from X to Y.

Return the value obtained by the formula $P(X,Y) = \int_{-\infty}^{Y} p(X,Y')dY'$.

Implements **PDF< std::vector< double > >** (p. 123).

3.5.3.3 double MvIndNormalPDF::evalMarginalCdf (double *x*, long *i*) const

Evaluate the ith marginal cdf of the distribution at x.

Return the value obtained by the formula $P_i(x) = \int_{-\infty}^{x} p_i(x')dx'$.

3.5.3.4 double MvIndNormalPDF::evalMarginalCdf (double *x*, double *y*, long *i*) const

Evaluate the ith marginal kernel cdf from x to y.

Return the value obtained by the formula $P_i(x,y) = \int_{-\infty}^{y} p_i(x,y')dy'$.

3.5.3.5 double MvIndNormalPDF::evalMarginalInvCdf (double *p*, long *i*) const

Evaluate the inverse cdf of the ith marginal distribution at probability p.

Return the value $P_i^{-1}(p)$.

3.5.3.6 double MvIndNormalPDF::evalMarginalInvCdf (double *p*, double *x*, long *i*) const

Evaluate the inverse kernel cdf from x of the ith marginal at probability p.

Return the value $P_i^{-1}(p,x)$.

3.5.3.7 double MvIndNormalPDF::evalMarginalPdf (double *x*, long *i*) const

Evaluate the ith marginal pdf of the distribution at x.

Return the value obtained by the formula $p_i(x) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(\frac{-(x-\mu_i)^2}{2\sigma_i^2}\right)$.

3.5.3.8 double MvIndNormalPDF::evalMarginalPdf (double *x*, double *y*, long *i*) const

Evaluate the ith marginal kernel pdf from x to y.

Return the value obtained by the formula $p_i(x,y) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(\frac{-(y-x)^2}{2\sigma_i^2}\right)$.

3.5.3.9 double MvIndNormalPDF::evalPdf (const std::vector< double > & *X*) const [virtual]

Evaluate the pdf at the vector X.

Return the value obtained by the formula $p(X) = \prod_{i=1}^{d} p(x_i)$ with $p(x_i) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(\frac{-(x_i-\mu_i)^2}{2\sigma_i^2}\right)$.

Implements **PDF< std::vector< double > >** (p. 123).

3.5.3.10 double MvIndNormalPDF::evalPdf (const std::vector< double > & *X*, const std::vector< double > & *Y*) const [virtual]

Evaluate the kernel pdf from X to Y.

Return the value obtained by the formula $p(X,Y) = \prod_{i=1}^{d} p(x_i,y_i)$ with $p(x_i,y_i) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(\frac{-(y_i-x_i)^2}{2\sigma_i^2}\right)$.

Implements **PDF< std::vector< double > >** (p. 123).

3.5.3.11 double MvIndNormalPDF::getMarginalSample (long *i*) const

Get a sample from the ith marginal distribution.

Use the inherited random number generator to return a Gaussian random variate distributed according to the ith marginal.

3.5.3.12 double MvIndNormalPDF::getMarginalSample (double *x*, long *i*) const

Get a sample from the ith kernel marginal distribution from x.

Use the inherited random number generator to return a Gaussian random variate distributed according to the ith marginal but with mean x.

3.5.3.13 std::vector< double > MvIndNormalPDF::getMeanVector ( ) const

Get the mean vector of the normal distribution.

Return a copy of the mean vector.

3.5.3.14 double MvIndNormalPDF::getMeanVectorComponent (long *i*) const

Get one component of the mean vector of the normal distribution.

Access the mean vector to return one element.

3.5.3.15 std::vector< double > MvIndNormalPDF::getSample ( ) const [virtual]

Get a sample vector from the distribution.

Use the inherited random number generator to return a Gaussian random vector.

Implements **PDF< std::vector< double > >** (p. 123).

3.5.3.16 std::vector< double > MvIndNormalPDF::getSample (const std::vector< double > & *X*) const [virtual]

Get a sample from the kernel distribution from X.

Use the inherited random number generator to return a Gaussian random vector with mean X.

Implements **PDF< std::vector< double > >** (p. 123).

3.5.3.17 std::vector< double > MvIndNormalPDF::getSample (const std::vector< double > & *X*, long *i*) const [virtual]

Get a sample from the kernel distribution by modifying only the ith component.

Use the inherited random number generator to return an updated sample vector with the ith component updated with the ith marginal distribution.

Implements **PDF< std::vector< double > >** (p. 123).

3.5.3.18 std::vector< double > MvIndNormalPDF::getStdDevVector ( ) const

Get the standard deviation vector of the normal distribution.

Return a copy of the standard deviation vector.

3.5.3.19 double MvIndNormalPDF::getStdDevVectorComponent (long *i*) const

Get one component of the standard deviation vector of the normal distribution.

Access the standard deviation vector to return one element.

3.5.3.20 void MvIndNormalPDF::setMeanVector (const std::vector< double > & *mean*)

Set the mean vector of the normal distribution.

Copy the argument vector into the mean vector.

3.5.3.21 void MvIndNormalPDF::setMeanVectorComponent (double *mean_i*, long *i*)

Set one component of the mean vector of the normal distribution.

Access the mean vector to change one element.

3.5.3.22 void MvIndNormalPDF::setStdDevVector (const std::vector< double > & *std_dev*)

Set the standard deviation vector of the normal distribution.

Copy the argument vector into the standard deviation vector.

3.5.3.23 void MvIndNormalPDF::setStdDevVectorComponent (double *std_dev_i*, long *i*)

Set one component of the standard deviation vector of the normal distribution.

Access the standard deviation vector to change one element.

The documentation for this class was generated from the following files:

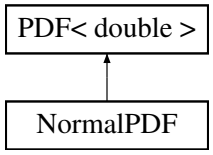
- MvIndNormalPDF.h
- MvIndNormalPDF.cpp

3.6 NormalPDF Class Reference

Inherited class representing a univariate Gaussian distribution.

`#include <NormalPDF.h>`

Inheritance diagram for NormalPDF:



Public Member Functions

- **NormalPDF** ()
  Constructor of the class.
- **NormalPDF** (double mean, double std_dev)
  Alternative constructor.
- virtual ~**NormalPDF** ()
  Destructor of the class.
- void **setMean** (double mean)
  Set the mean of the normal distribution.
- double **getMean** () const
  Get the mean of the normal distribution.
- void **setStdDev** (double std_dev)
  Set the standard deviation of the normal distribution.
- double **getStdDev** () const
  Get the standard deviation of the normal distribution.
- virtual double **evalPdf** (const double &x) const
  Evaluate the pdf at point x.
- virtual double **evalCdf** (const double &x) const
  Evaluate the cdf at point x.
- virtual double **evalInvCdf** (double p) const
  Evaluate the inverse cdf at probability p.
- virtual double **getSample** () const
  Get a sample from the distribution.
- virtual double **evalPdf** (const double &x, const double &y) const
  Evaluate the kernel pdf from x to y.
- virtual double **evalCdf** (const double &x, const double &y) const
  Evaluate the kernel cdf from x to y.
- virtual double **evalInvCdf** (double p, const double &x) const
  Evaluate the inverse kernel cdf from x.
- virtual double **getSample** (const double &x) const
  Get a sample from a kernel from x.

Protected Attributes

- double **mu**

  Mean of the normal distribution.
- double **sigma**

  Standard deviation of the normal distribution.

3.6.1 Detailed Description

Inherited class representing a univariate Gaussian distribution.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 NormalPDF::NormalPDF ( )

Constructor of the class.

Default constructor, set the mean to 0 and the standard deviation to 1.

3.6.2.2 NormalPDF::NormalPDF (double *mean*, double *std_dev*)

Alternative constructor.

Enable the user to define the mean and the standard deviation.

3.6.3 Member Function Documentation

3.6.3.1 double NormalPDF::evalCdf (const double & *x*) const [virtual]

Evaluate the cdf at point x.

Return the value obtained by the formula $P(x) = \int_{-\infty}^{x} p(x')dx'$.

Implements **PDF**< **double** > (p. 123).

3.6.3.2 double NormalPDF::evalCdf (const double & *x*, const double & *y*) const [virtual]

Evaluate the kernel cdf from x to y.

Return the value obtained by the formula $P(x,y) = \int_{-\infty}^{y} p(x,y')dy'$.

Implements **PDF**< **double** > (p. 123).

3.6.3.3 double NormalPDF::evalInvCdf (double *p*) const [virtual]

Evaluate the inverse cdf at probability p.

Return the value $P^{-1}(p)$.

Implements **PDF**< **double** > (p. 123).

3.6.3.4 double NormalPDF::evalInvCdf (double *p*, const double & *x*) const [virtual]

Evaluate the inverse kernel cdf from x.

Return the value $P^{-1}(p,x)$.

Implements **PDF< double >** (p. 123).

3.6.3.5 double NormalPDF::evalPdf (const double & *x*) const [virtual]

Evaluate the pdf at point x.

Return the value obtained by the formula $p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)$.

Implements **PDF< double >** (p. 123).

3.6.3.6 double NormalPDF::evalPdf (const double & *x*, const double & *y*) const [virtual]

Evaluate the kernel pdf from x to y.

Return the value obtained by the formula $p(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(y-x)^2}{2\sigma^2}\right)$.

Implements **PDF< double >** (p. 123).

3.6.3.7 double NormalPDF::getSample ( ) const [virtual]

Get a sample from the distribution.

Use the inherited random number generator to return a Gaussian random variate.

Implements **PDF< double >** (p. 123).

3.6.3.8 double NormalPDF::getSample (const double & *x*) const [virtual]

Get a sample from a kernel from x.

Use the inherited random number generator to return a Gaussian random variate with mean x.

Implements **PDF< double >** (p. 123).

The documentation for this class was generated from the following files:

- NormalPDF.h
- NormalPDF.cpp

3.7 Particle< T > Class Template Reference

Base template class for a particle in SMC algorithms.

`#include <Particle.h>`

Public Member Functions

- **Particle** ()

Constructor of the class.

- **Particle** (const T &)

  Alternative constructor.
- virtual ~**Particle** ()

  Destructor of the class.
- virtual const **Particle**< T > & **operator=** (const **Particle**< T > &)

  Overloaded assignment operator.
- void **setLogWeight** (double logw)

  Set the logarithm of the importance weight.
- void **addToLogWeight** (double increment)

  Add a quantity to the logarithm of the importance weight.
- double **getLogWeight** () const

  Get the logarithm of the importance weight.
- double **getWeight** () const

  Get the importance weight.
- void **setFunValue** (double new_val)

  Set the value of some expensive function for the current particle.
- double **getFunValue** () const

  Get the value of some expensive function for the current particle.
- void **setComponents** (const T &)

  Set the components of the particle.
- T **getComponents** () const

  Get the components of the particle.

Protected Attributes

- double **log_weight**

  Logarithm of the importance weight.
- double **fun_value**

  Value of some expensive function for this particle.
- T **components**

  Components of the particle.

3.7.1 Detailed Description

template<typename T>class Particle< T >

Base template class for a particle in SMC algorithms.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 template<typename T > Particle< T >::Particle ( )

Constructor of the class.

Initialize the log-weight to default values.

3.7.2.2 template<typename T > Particle< T >::Particle (const T & *init*)

Alternative constructor.

Initialize the components to defined values.

3.7.3 Member Function Documentation

3.7.3.1 template<typename T > T Particle< T >::getComponents ( ) const

Get the components of the particle.

Return a copy of the components vector.

3.7.3.2 template<typename T > const Particle< T > & Particle< T >::operator= (const Particle< T > & *rhs*) [virtual]

Overloaded assignment operator.

Copy an entire particle.

3.7.3.3 template<typename T > void Particle< T >::setComponents (const T & *new_comps*)

Set the components of the particle.

Copy the argument vector in the components vector.

The documentation for this class was generated from the following file:

- Particle.h

3.8 ParticleSet< T, np > Class Template Reference

Template class representing a set of np particles.

`#include <ParticleSet.h>`

Public Member Functions

- **ParticleSet** ()
  Constructor of the class.
- **ParticleSet** (const T &)
  Alternative constructor.
- virtual ~**ParticleSet** ()
  Destructor of the class.
- **Particle**< T > **operator[]** (long i) const
  Get the ith particle in the set, const return.
- **Particle**< T > & **operator[]** (long i)
  Get the ith particle in the set, non const return.
- double **computeEss** ()

Compute the effective sample size.

- void **normalizeWeights** ()

  Normalize the weights.

- void **uniformizeWeights** ()

  Uniformize the weights.

- void **resampleParticles** ()

  Resample the particles.

- void **setRandNumGen** (gsl_rng *r)

  Set the random number generator.

Protected Attributes

- gsl_rng * **rand_gen**

  Pointer to an instance of a random number generator.

- std::vector< **Particle**< T > > **particles**

  Vector containing the particles.

3.8.1 Detailed Description

template<typename T, long np>class ParticleSet< T, np >

Template class representing a set of np particles.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 template<typename T , long np> ParticleSet< T, np >::ParticleSet ( )

Constructor of the class.

Initialize the class with default values.

3.8.2.2 template<typename T, long np> ParticleSet< T, np >::ParticleSet (const T & *init*)

Alternative constructor.

Initialize the class with specified values for the particles.

3.8.3 Member Function Documentation

3.8.3.1 template<typename T , long np> double ParticleSet< T, np >::computeEss ( )

Compute the effective sample size.

Normalize the weights and return the value computed by the formula $ESS = 1/\sum_{k=1}^{np} W_k^2$.

3.8.3.2 template<typename T , long np> void ParticleSet< T, np >::normalizeWeights ( )

Normalize the weights.

Normalization of the log-weights.

3.8.3.3 template<typename T , long np> void ParticleSet< T, np >::resampleParticles ( )

Resample the particles.

Systematic resampling according to the weights of the particles.

3.8.3.4 template<typename T , long np> void ParticleSet< T, np >::uniformizeWeights ( )

Uniformize the weights.

Set all the log-weights to log(1/np)

The documentation for this class was generated from the following file:

- ParticleSet.h

3.9 PDF< T > Class Template Reference

Abstract template base class for probability distributions.

`#include <PDF.h>`

Public Member Functions

- **PDF** ()
  Constructor of the class.
- virtual ~**PDF** ()
  Destructor of the class.
- virtual double **evalPdf** (const T &) const =0
  Evaluate the pdf.
- virtual double **evalCdf** (const T &) const =0
  Evaluate the cdf.
- virtual T **evalInvCdf** (double) const =0
  Evaluate the inverse cdf.
- virtual T **getSample** () const =0
  Get a sample from the distribution.
- virtual double **evalPdf** (const T &, const T &) const =0
  Evaluate the pdf, kernel version.
- virtual double **evalCdf** (const T &, const T &) const =0
  Evaluate the cdf, kernel version.
- virtual T **evalInvCdf** (double, const T &) const =0
  Evaluate the inverse cdf, kernel version.
- virtual T **getSample** (const T &) const =0
  Get a sample from the distribution, kernel version.
- virtual T **getSample** (const T &, long) const =0
  Modify only one component.
- virtual void **setRandNumGen** (gsl_rng *r)
  Set the random number generator.

Protected Attributes

- gsl_rng * **rand_gen**

  Pointer to an instance of a random number generator.

3.9.1 Detailed Description

template<typename T>class PDF< T >

Abstract template base class for probability distributions.

The documentation for this class was generated from the following file:

- PDF.h

3.10 PDFSequence< T > Class Template Reference

Abstract template base class for SMC sequence of probability distributions.

`#include <PDFSequence.h>`

Public Member Functions

- **PDFSequence** ()

  Constructor of the class.
- virtual ~**PDFSequence** ()

  Destructor of the class.
- void **setCurrentSeqParam** (double seqparam)

  Set the current sequence parameter.
- double **getCurrentSeqParam** ()

  Get the current sequence parameter.
- void **setNextSeqParam** (double seqparam)

  Set the next sequence parameter.
- double **getNextSeqParam** ()

  Get the next sequence parameter.
- virtual double **evalCurrentPdf** (const T &) const =0

  Evaluate the current pdf in the sequence.
- virtual double **evalCurrentPdf** (const T &, double) const =0

  Evaluate the current pdf in the sequence using an expensive function.
- virtual double **evalNextPdf** (const T &) const =0

  Evaluate the next pdf in the sequence.
- virtual double **evalNextPdf** (const T &, double) const =0

  Evaluate the next pdf in the sequence using an expensive function.
- virtual double **evalFun** (const T &) const =0

  Return the value of the expensive function used in the pdf evaluations.

Protected Attributes

- double **seqparam_current**

  Current sequence parameter.
- double **seqparam_next**

  Next sequence parameter.

3.10.1 Detailed Description

template<typename T>class PDFSequence< T >

Abstract template base class for SMC sequence of probability distributions.

The documentation for this class was generated from the following file:

- PDFSequence.h

3.11 SMCSampler< T, num_particles > Class Template Reference

Base class for a Sequential Monte Carlo sampler.

`#include <SMCSampler.h>`

Public Member Functions

- **SMCSampler** ()

  Constructor of the class.
- virtual ~**SMCSampler** ()

  Destructor of the class.
- long **getNumParticles** () const

  Get the number of particles.
- void **setSeqParamStart** (double seqparam)

  Set the initial sequence parameter.
- double **getSeqParamStart** () const

  Get the initial sequence paramater.
- void **setSeqParamEnd** (double seqparam)

  Set the final sequence parameter.
- double **getSeqParamEnd** () const

  Get the final sequence parameter.
- void **setSeqParamCurrent** (double seqparam)

  Set the current sequence parameter.
- double **getSeqParamCurrent** () const

  Get the current sequence parameter.
- void **setSeqParamNext** (double seqparam)

  Set the next sequence parameter.
- double **getSeqParamNext** () const

  Get the next sequence parameter.
- void **setEssFactorThreshold** (double essratio)

Set the ESS ratio threshold for the adaptivity.

- double **getEssFactorThreshold** () const

  Get the ESS ratio threshold for the adaptivity.

- void **setEssFactorMargin** (double essmargin)

  Set the error margin for the ESS ratio threshold.

- double **getEssFactorMargin** () const

  Get the error margin for the ESS ratio threshold.

- void **setEssResamplingRatio** (double essratio)

  Set the resampling ESS ratio.

- double **getEssResamplingRatio** () const

  Get the resampling ESS ratio.

- void **setNumMcmcMoves** (long nmoves)

  Set the number of McMC updates per particle per iteration.

- long **getNumMcmcMoves** () const

  Get the number of McMC updates per particle per iteration.

- void **setNumAccepted** (long naccept)

  Set the number of accepted McMC moves.

- void **addToNumAccepted** (long naccept)

  Add some number to the number of accepted McMC moves.

- long **getNumAccepted** () const

  Get the number of accepted McMC moves.

- void **setNumResamplings** (long nresample)

  Set the number of performed resamplings.

- void **addNumResamplings** ()

  Add one to the number of performed resamplings.

- long **getNumResamplings** () const

  Get the number of performed resamplings.

- void **setNumSmcIterations** (long numiter)

  Set the number of SMC iterations.

- void **addNumSmcIterations** ()

  Add one to the number of SMC iterations.

- long **getNumSmcIterations** () const

  Get the number of SMC iterations.

- void **setMcmcMove** (**MCMCUpdate**< T > &)

  Set the McMC update scheme to be used.

- void **setMcmcTransPdf** (**PDF**< T > &)

  Set the transition pdf to be used in the McMC.

- void **setPdfSequence** (**PDFSequence**< T > &)

  Set the sequence of probability distributions.

- void **setRandNumGen** (gsl_rng *r)

  Set the random number generator.

- void **setSeedNumber** (long seed)

  Set the seed number for the random number generator.

- virtual void **scanInputParameters** ()

  Scan input date from a text file.

- virtual void **initializeSampler** ()

  Initialize the particles.

- virtual void **performSmcLoop** ()

  Performs the SMC loop.
- virtual void **performSmcLoopWithReport** ()

  Performs the SMC loop writing an intermediate report.
- virtual void **outputReport** ()

  Output results in a report file.
- void **findNextSeqParam** ()

  Determine the next sequence parameter.
- void **updateWeights** ()

  Update the weights of the particles.
- void **resampleParticles** ()

  Resample the particles if necessary.
- virtual void **moveParticles** ()

  Move all the particles.

Protected Attributes

- double **seqparam_start**

  Initial pdf sequence parameter.
- double **seqparam_end**

  Final pdf sequence parameter.
- double **seqparam_current**

  Current pdf sequence parameter.
- double **seqparam_next**

  Next pdf sequence parameter.
- double **ess_factor_threshold**

  ESS ratio threshold for the adaptivity.
- double **ess_factor_margin**

  Error margin for the ESS ratio threshold.
- double **ess_resampling_ratio**

  Resampling ESS ratio.
- long **num_mcmc_moves**

  Number of McMC updates per particle per iteration.
- long **num_accepted**

  Number of accepted McMC moves.
- long **num_resamplings**

  Number of performed resamplings.
- long **num_smc_iterations**

  Number of SMC iterations.
- **MCMCUpdate**< T > * **mcmc_move**

  Pointer to the McMC update used to move the particles.
- **PDF**< T > * **mcmc_transition_pdf**

  Pointer to the transition pdf in the McMC moves.
- **PDFSequence**< T > * **pdf_sequence**

  Pointer to the parametrized sequence of probability distributions.
- **ParticleSet**< T, num_particles > **particle_set**

  Set of particles.

- gsl_rng * **rand_gen**

  Pointer to an instance of a random number generator.
- long **seed_number**

  Seed number for the random number generator.

3.11.1 Detailed Description

template<typename T, long num_particles>class SMCSampler< T, num_particles >

Base class for a Sequential Monte Carlo sampler.

3.11.2 Constructor & Destructor Documentation

3.11.2.1 template<typename T , long num_particles> SMCSampler< T, num_particles >::SMCSampler ( )

Constructor of the class.

Default constructor, set data members to default values.

3.11.3 Member Function Documentation

3.11.3.1 template<typename T , long num_particles> void SMCSampler< T, num_particles >::findNextSeqParam ( )

Determine the next sequence parameter.

Adaptivity step.

3.11.3.2 template<typename T , long num_particles> void SMCSampler< T, num_particles >::initializeSampler ( ) [virtual]

Initialize the particles.

To be overriden to initialize the particles values.

3.11.3.3 template<typename T , long num_particles> void SMCSampler< T, num_particles >::moveParticles ( ) [virtual]

Move all the particles.

Rejuvenating step.

3.11.3.4 template<typename T , long num_particles> void SMCSampler< T, num_particles >::outputReport ( ) [virtual]

Output results in a report file.

Print relevant data in a file.

3.11.3.5 template<typename T , long num_particles> void SMCSampler< T, num_particles >::performSmcLoop ( ) [virtual]

Performs the SMC loop.

Main member function of the class.

3.11.3.6 template<typename T , long num_particles> void SMCSampler< T, num_particles >::performSmcLoopWithReport ( ) [virtual]

Performs the SMC loop writing an intermediate report.

Main member function of the class, writes intermediate data.

3.11.3.7 template<typename T , long num_particles> void SMCSampler< T, num_particles >::resampleParticles ( )

Resample the particles if necessary.

Resampling step.

3.11.3.8 template<typename T , long num_particles> void SMCSampler< T, num_particles >::scanInputParameters ( ) [virtual]

Scan input date from a text file.

Read parameters and modify the corresponding data members.

3.11.3.9 template<typename T , long num_particles> void SMCSampler< T, num_particles >::setMcmcMove (MCMCUpdate< T > & *mcmc*)

Set the McMC update scheme to be used.

Set the pointer to the address of the argument.

3.11.3.10 template<typename T , long num_particles> void SMCSampler< T, num_particles >::setMcmcTransPdf (PDF< T > & *transpdf*)

Set the transition pdf to be used in the McMC.

Set the pointer to the address of the argument.

3.11.3.11 template<typename T , long num_particles> void SMCSampler< T, num_particles >::setNumMcmcMoves (long *nmoves*)

Set the number of McMC updates per particle per iteration.

Also modifies the McMC update object.

3.11.3.12 template<typename T , long num_particles> void SMCSampler< T, num_particles >::setPdfSequence (PDFSequence< T > & *pdfseq*)

Set the sequence of probability distributions.

Set the pointer to the address of the argument.

3.11.3.13 template<typename T , long num_particles> void SMCSampler< T, num_particles >::setRandNumGen (gsl_rng * r)

Set the random number generator.

Set the random number generator in the sub-objects.

3.11.3.14 template<typename T , long num_particles> void SMCSampler< T, num_particles >::setSeqParamCurrent (double *seqparam*)

Set the current sequence parameter.

Also modifies the pdf sequence object.

3.11.3.15 template<typename T , long num_particles> void SMCSampler< T, num_particles >::setSeqParamNext (double *seqparam*)

Set the next sequence parameter.

Also modifies the pdf sequence object.

3.11.3.16 template<typename T , long num_particles> void SMCSampler< T, num_particles >::updateWeights ( )

Update the weights of the particles.

Reweighing step.

The documentation for this class was generated from the following file:

- SMCSampler.h

B.2 Example program

We will describe in this section a simple program to sample from a probability distribution using `SMCLib`. The only coding prerequisites are:

- A class representing the target sequence of densities using inheritance from `PDFSequence`, here this class was defined in the header `MixtureGaussianSequence.h`. All pure virtual functions need to be overloaded. In our example, we seek to sample from a univariate distribution representing a mixture of three Gaussian distributions. Hence, the inheritance relation was written as:
  `class MixtureGaussianSequence : public PDFSequence<double>`.
- A class representing a `SMCSampler` object adapted to our problem, using once again inheritance. Only the functions initializing the sampler (`initializeSampler()`) and writing the final report (`outputReport()`) need to be overloaded. Using 1,000 particles, the inheritance relation can be written as:
  `class SMCMixture : public SMCSampler<double,1000>`.

We give in the following an example of the required `main.cpp` file to run the sampler, followed by a concise explanation on its contents. The `#include` directives tell the C++ preprocessor to be aware of the contents of specific headers: the `NormalPDF.h` header corresponding to the chosen transition probability distribution of the MCMC kernel involved in the rejuvenation step in SMC, as well as the headers corresponding to the two previously discussed classes. All other required headers, such as the ones corresponding to the C++ Standard Library and the GNU Scientific Library, are included in `#include` directives written in `SMCLib` headers such as `SMCSampler.h`.

```
#include "NormalPDF.h"
#include "MixtureGaussianSequence.h"
#include "SMCMixture.h"

using namespace std;

int main()
{
  gsl_rng * r = gsl_rng_alloc(gsl_rng_default);

  MixtureGaussianSequence mytargetpdf(-30,0.5,0,0.5,30,0.5);

  NormalPDF mytransitionpdf(0,5);
  mytransitionpdf.setRandNumGen(r);

  MCMCUpdate<double> mymcmc;
  SMCMixture mysmc;

  mysmc.setMcmcMove(mymcmc);
  mysmc.setMcmcTransPdf(mytransitionpdf);
  mysmc.setPdfSequence(mytargetpdf);
  mysmc.setNumMcmcMoves(5);

  mysmc.setRandNumGen(r);
  mysmc.setSeedNumber(0);
```

```
  mysmc.performSmcLoopWithReport();
  mysmc.outputReport();

  gsl_rng_free(r);
  return 0;
}
```

In the `main()` function, the first line allocates a new random number generator using the GNU Scientific Library. Then we declare the target density `mytargetpdf`, our transition distribution `mytransitionpdf` and we link the latter to the random number generator `r`. The following step is to declare the MCMC kernel we use, `mymcmc`, as well as an instance of our SMC sampler, `mysmc`. The subsequent block links the declared MCMC kernel `mymcmc`, associated transition distribution `mytransitionpdf` and the target density `mytargetpdf` to the sampler object `mysmc`. We then initialize the SMC sampler random number generator, before running the sampler itself and generating reports of both the sampling process and its output. The `main()` function terminates by clearing the allocated random number generator `r` and returning the value 0.

BIBLIOGRAPHY

[1] Pierre-Antoine Absil, Robert Mahony, and Rodolphe Sepulchre. Riemannian geometry of grassmann manifolds with a view on algorithmic computation. *Acta Applicandae Mathematicae*, 80:199–220, 2004.

[2] Pierre-Antoine Absil, Robert Mahony, and Rodolphe Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, 2009.

[3] B.O. Almroth, P. Stern, and F.A. Brogan. Automatic choice of global shape functions in structural analysis. *AIAA Journal*, 16(5):525–528, 1978.

[4] Shun'ichi Amari and Hiroshi Nagaoka. *Methods of information geometry*. Translations of mathematical monographs. American Mathematical Society, 2000.

[5] David Amsallem, Julien Cortial, Kevin Carlberg, and Charbel Farhat. A method for interpolating on manifolds structural dynamics reduced-order models. *International Journal for Numerical Methods in Engineering*, 80:1241–1258, 2009.

[6] David Amsallem and Charbel Farhat. Interpolation method for adapting reduced-order models and application to aeroelasticity. *AIAA Journal*, 46(7), 2008.

[7] B.A. Amzal, F. Bois, E. Parent, and C.P. Robert. Bayesian-optimal design via interacting particle systems. *Journal of the American Statistical Association*, 101(474):773–785, 2006.

[8] Christophe Andrieu, Nando de Freitas, Arnaud Doucet, and Michael I. Jordan. An introduction to mcmc for machine learning. *Machine Learning*, 50:5–43, 2003.

[9] A.C. Atkinson, A.N. Donev, and R.D. Tobias. *Optimum Experimental Design, with SAS*. Oxford Statistical Science Series. Oxford University Press, 2007.

[10] Evgeni Begelfor and Michael Werman. Affine invariance revisited. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2006.

[11] M.P. Bendsoe and O. Sigmund. Material interpolation schemes in topology optimization. *Archive of Applied Mechanics*, 69(9-10):635 – 654, 1999.

[12] N.F. Berk. Scattering properties of leveled-wave model for random morphologies. *Physical Review A*, 44(9):5069–5079, 1991.

[13] J. Besag. Markov chain monte carlo for statistical inference. Technical report, University of Washington, Center for Statistics and the Social Sciences, 2001.

[14] Julian Besag, Peter Green, David Higdon, and Kerrie Mengersen. Bayesian computation and stochastic systems. *Statistical Science*, 10(1):3–66, 1995.

[15] R.A. Bialecki, A. Kassab, and A. Fic. Proper orthogonal decomposition and modal analysis for acceleration of transient fem analysis. *International Journal for Numerical Methods in Engineering*, 62:774–797, 2005.

[16] S. Boyaval, C. Le Bris, Y. Maday, N.C. Nguyen, and A.T. Patera. A reduced basis approach for variational problems with stochastic parameters: Application to heat conduction with variable robin coefficient. *Computer Methods in Applied Mechanics and Engineering*, 198:3187–3206, 2009.

[17] T. Bui-Thanh, Karen Willcox, and Omar Ghattas. Model reduction for large-scale systems with high-dimensional parametric input space. *SIAM Journal of Scientific Computing*, 30(6):3270–3288, 2008.

[18] John Burkardt, Qiang Du, and Max Gunzburger. Reduced order modeling of complex systems. In *NA03 Dundee*, 2003.

[19] John Burkardt, Max Gunzburger, and Hyung-Chun Lee. Centroidal voronoi tesselation-based reduced-order modeling of complex systems. *SIAM Journal of Scientific Computing*, 28(2):459–484, 2006.

[20] Olivier Cappé, Simon J. Godsill, and Eric Moulines. An overview of existing methods and recent advances in sequential monte carlo. *Proceedings of the IEEE*, 95(5), 2007.

[21] Kathryn Chaloner and Isabella Verdinelli. Bayesian experimental design: A review. *Statistical Science*, 10(3):273–304, 1995.

[22] D.C. Charmpis, G.I. Schueller, and M.F. Pellissetti. The need for linking micromechanics of materials with stochastic finite elements: A challenge for materials science. *Computational Materials Science*, 41(1):27 – 37, 2007.

[23] Nicolas Chopin. A sequential particle filter method for static models. *Biometrika*, 89(3):539–551, 2002.

[24] Nicolas Chopin. Central limit theorem for sequential monte carlo methods and its applications to bayesian inference. *The Annals of Statistics*, 32(6):2385–2411, 2004.

[25] R.W. Clough. The finite element method in plane stress analysis. In *Proceedings, Second ASCE Conference on Electronic Computation*, 1960.

[26] Joris Degroote, Jan Vierendeels, and Karen Willcox. Interpolation among reduced-order matrices to obtain parameterized models for design, optimization and probabilistic analysis. *International Journal for Numerical Methods in Fluids*, 63:207–230, 2010.

[27] Pierre del Moral. *Feynman-Kac Formulae, Genealogical and Interacting Particle Systems with Applications*. Springer, 2004.

[28] Pierre del Moral, Arnaud Doucet, and Ajay Jasra. Sequential monte carlo samplers. *Journal of the Royal Statistical Society Series B*, 68:411–436, 2006.

[29] Pierre del Moral, Arnaud Doucet, and Ajay Jasra. Sequential monte carlo for bayesian computation. In J.M. Bernardo, M.J. Bayarri, J.O. Berger, A.P. Dawid, D. Heckerman, A.F.M. Smith, and M. West, editors, *Bayesian Statistics*, volume 8. Oxford University Press, 2007.

[30] Manfredo P. do Carmo. *Riemannian Geometry*. Birkhäuser, 1992.

[31] Arnaud Doucet, Nando de Freitas, and N. Gordon. *Sequential Monte Carlo in Practice*. Springer, 2001.

[32] Arnaud Doucet, Simon Godsill, and Christophe Andrieu. On sequential monte carlo sampling methods for bayesian filtering. *Statistics and Computing*, 10:197–208, 2000.

[33] Arnaud Doucet, Simon J. Godsill, and Christian P. Robert. Marginal maximum a posteriori estimation using markov chain monte carlo. *Statistics and Computing*, 12:77–84, 2002.

[34] Qiang Du, Vance Faber, and Max Gunzburger. Centroidal voronoi tesselations: Applications and algorithms. *SIAM Review*, 41(4):637–676, 1999.

[35] S. Duane, A.D. Kennedy, B.J. Pendleton, and D. Roweth. Hybrid monte carlo. *Physics Letters B*, 195(2):216–222, 1987.

[36] Alan Edelman, Tomás A. Arias, and Steven T. Smith. The geometry of algorithms with orthogonality constraints. *SIAM Journal of Matrix Analysis and Applications*, 20(2):303–353, 1998.

[37] Valerii V. Fedorov and Peter Hackl. *Model-Oriented Design of Experiments*. Lecture Notes in Statistics. Springer, 1997.

[38] D. Galbally, K. Fidkowski, K. Willcox, and O. Ghattas. Non-linear model reduction for uncertainty quantification in large-scale inverse problems. *International Journal for Numerical Methods in Engineering*, 2009.

[39] Andrew Gelman, John B. Carlin, and Hal S. Stern. *Bayesian Data Analysis*. Chapman and Hall/CRC, 2003.

[40] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, 1984.

[41] A. Gersborg-Hansen, M.P. Bendsoe, and O. Sigmund. Topology optimization of heat conduction problems using the finite volume method. *Structural and Multidisciplinary Optimization*, 31(4):251 – 259, 2006.

[42] R. Ghanem and P. Spanos. *Stochastic Finite Elements: A Spectral Approach*. Springer-Verlag, 1991.

[43] L.V. Gibiansky and O. Sigmund. Multiphase composites with extremal bulk modulus. *Journal of the Mechanics and Physics of Solids*, 48(3):461 – 498, 2000.

[44] Mark Girolami and Ben Calderhead. Riemann manifold langevin and hamiltonian monte carlo methods. *Journal of the Royal Statistical Society Series B*, 73:123–214, 2011.

[45] N.J. Gordon, D.J. Salmond, and A.F.M. Smith. Novel approach to nonlinear non-gaussian bayesian state estimation. In *IEEE Proceedings on Radar and Signal Processing*, volume 140, pages 107–113, 1993.

[46] M.A. Grepl, Y. Maday, N.C. Nguyen, and A.T. Patera. Efficient reduced-basis treatment of nonaffine and nonlinear partial differential equations. *Mathematical Modelling and Numerical Analysis*, 41(3):575–605, 2007.

[47] J.K. Guest and T. Igusa. Structural optimization under uncertain loads and nodal locations. *Computer Methods in Applied Mechanics and Engineering*, 198(1):116 – 124, 2008.

[48] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[49] D. Higdon, H. Lee, and C. Holloman. Markov chain monte carlo based approaches for inference in computationally intensive inverse problems. In J.M. Bernardo, M.J. Bayarri, J.O. Berger, A.P. Dawid, D. Heckerman, A.F.M. Smith, and M. West, editors, *Bayesian Statistics*, volume 7. Oxford University Press, 2003.

[50] Philip Holmes, John L. Lumley, and Gal Berkooz. *Turbulence, Coherent Structures, Dynamical Systems and Symmetry*. Cambridge University Press, 1996.

[51] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24, 1933.

[52] Adam M. Johansen. Smctc: Sequential monte carlo in c++. *Journal of Statistical Software*, 30(6), 2009.

[53] Adam M. Johansen, Arnaud Doucet, and Manuel Davy. Particle methods for maximum likelihood estimation in latent variable models. *Statistics and Computing*, 18:47–57, 2008.

[54] Jari Kaipio and Erkki Somersalo. *Statistical and Computational Inverse Problems*. Springer, 2004.

[55] Robert E. Kass and Larry Wasserman. The selection of prior distributions by formal rules. *Journal of the American Statistical Association*, 91(435), 1996.

[56] M.C. Kennedy and A. O'Hagan. Predicting the output from a complex computer code when fast approximations are available. *Biometrika*, 87(1):1–13, 2000.

[57] Phaedon-Stelios Koutsourelakis. Accurate uncertainty quantification using inaccurate models. *SIAM Journal of Scientific Computing*, 31(5):3274–3300, 2009.

[58] Phaedon-Stelios Koutsourelakis. A multi-resolution, non-parametric, bayesian framework for identification of spatially-varying model parameters. *Journal of Computational Physics*, 228:6184–6211, 2009.

[59] P.S. Koutsourelakis. Probabilistic characterization and simulation of multi-phase random media. *Probabilistic Engineering Mechanics*, 21(3):227–234, 2006.

[60] P.S. Koutsourelakis and G. Deodatis. Simulation of binary random processes with applications to two-phase random media. *ASCE Journal of Engineering Mechanics*, 131(4):397–412, 2005.

[61] H. Kück, N. de Freitas, and A. Doucet. SMC Samplers for Bayesian Optimal Nonlinear Design. In *Nonlinear Statistical Signal Processing Workshop (NSSPW)*, 2006.

[62] Chad Lieberman, Karen Willcox, and Omar Ghattas. Parameter and state model reduction for large-scale statistical inverse problems. *SIAM Journal of Scientific Computing*, 32(5):2523–2542, 2010.

[63] Jun S. Liu. *Monte Carlo Strategies in Scientific Computing*. Springer, 2001.

[64] Xiang Ma and Nicholas Zabaras. An efficient bayesian inference approach to inverse problems based on an adaptive sparse grid collocation method. *Inverse Problems*, 25, 2009.

[65] S.N. MacEachern, M. Clyde, and J.S. Liu. Sequential importance sampling for nonparametric bayes models: The next generation. *The Canadian Journal of Statistics / La Revue Canadienne de Statistique*, 27(2):251–267, 1998.

[66] L. Machiels, Y. Maday, I.B. Oliveira, and A.T. Patera. Output bounds for reduced-basis approximations of symmetric positive definite eigenvalue problems. *Numerical Analysis*, 331(1):153–158, 2000.

[67] Youssef Marzouk and Dongbin Xiu. A stochastic collocation approach to bayesian inference in inverse problems. *Communications in Computational Physics*, 6(4):826–847, 2009.

[68] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, and Augusta H. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[69] Scott Meyers. *Effective C++: 55 Specific Ways to Improve your Programs and Designs*. Addison-Wesley Professional, 2005.

[70] Peter Müller. Simulation-based optimal design. In J.M. Bernardo, J.O. Berger, A.P. Dawid, and A.F.M. Smith, editors, *Bayesian Statistics*, volume 6. Oxford University Press, 1999.

[71] Ahmed K. Noor. Recent advances in reduction methods for nonlinear problems. *Computers and Structures*, 13:31–44, 1981.

[72] A.K. Noor and J.M. Peters. Reduced basis technique for nonlinear analysis of structures. *AIAA Journal*, 18(4):455–462, 1980.

[73] J.T. Oden and S. Prudhomme. Control of modeling error in calibration and validation processes for predictive stochastic models. *International Journal for Numerical Methods in Engineering*, 87:262–272, 2011.

[74] Z. Ostrowski. *Application of Proper Orthogonal Decomposition to the Solution of Inverse Problems*. PhD thesis, Silesian University of Technology, Gliwice, Poland, 2006.

[75] Z. Ostrowski, R.A. Bialecki, and A.J. Kassab. Estimation of constant thermal conductivity by use of proper orthogonal decomposition. *Computational Mechanics*, 37:52–59, 2005.

[76] Karl Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2, 1901.

[77] Xavier Pennec. Intrinsic statistics on riemannian manifolds: Basic tools for geometric measurements. *Journal of Mathematical Imaging and Vision*, 25:127–154, 2006.

[78] Xavier Pennec, Pierre Fillard, and Nicholas Ayache. A riemannian framework for tensor computing. *International Journal of Computer Vision*, 66(1):41–66, 2006.

[79] T.A. Porsching. Estimation of the error in the reduced basis method solution of nonlinear equations. *Mathematics of Computation*, 45(172):487–496, 1985.

[80] T.A. Porsching and M.L. Lee. The reduced basis method for initial value problems. *SIAM Journal on Numerical Analysis*, 24(6):1277–1287, 1987.

[81] C. Prudhomme, D.V. Rovas, K. Veroy, L. Machiels, Y. Maday, A.T. Patera, and G. Turinici. Reliable real-time solution of parametrized partial differential equations: Reduced-basis output bound methods. *Journal of Fluids Engineering*, 124, 2002.

[82] Christian P. Robert. *The Bayesian Choice*. Springer, 2001.

[83] Christian P. Robert and George Casella. *Monte Carlo Statistical Methods*. Springer, 2004.

[84] A.P. Roberts and M. Teubner. Transport properties of heterogeneous materials derived from gaussian random fields: Bounds and simulation. *Physical Review E*, 51(5):4141–4153, 1995.

[85] G.O. Roberts and J.S. Rosenthal. Optimal scaling for various metropolis-hastings algorithms. *Statistical Science*, 16:351–367, 2001.

[86] A. Romkes, J.T. Oden, and K. Vemaganti. Multi-scale goal-oriented adaptive modeling of random heterogeneous materials. *Mechanics of Materials*, 38(8-10):859 – 872, 2006.

[87] G. Rozza, D.B.P. Huynh, and A.T. Patera. Reduced basis approximation and a posteriori error estimation for affinely parametrized elliptic coercive partial differential equations. *Archives of Computational Methods in Engineering*, 15:229–275, 2008.

[88] Jerome Sacks, William J. Welch, Toby J. Mitchell, and Henry P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4(4):409–423, 1989.

[89] S. Sankaran and N. Zabaras. A maximum entropy approach for property prediction of random microstructures. *Acta Materialia*, pages 2265–2276, 2006.

[90] G.I. Schueller and H.A. Jensen. Computational methods in optimization considering uncertainties - an overview. *Computer Methods in Applied Mechanics and Engineering*, 198(1):2 – 13, 2008.

[91] Masanobu Shinozuka and George Deodatis. Simulation of multi-dimensional gaussian stochastic fields by spectral representation. *ASME Applied Mechanics Reviews*, 49(1), 1996.

[92] O. Sigmund. Materials with prescribed constitutive parameters - an inverse homogenization problem. *International Journal of Solids and Structures*, 31(17):2313 – 2329, 1994.

[93] O. Sigmund. Tailoring materials for specific needs. *Journal of Intelligent Material Systems and Structures*, 5(6):736 – 742, 1994.

[94] O. Sigmund. Tailoring materials with prescribed elastic properties. *Mechanics of Materials*, 20(4):351 – 368, 1995.

[95] O. Sigmund. On the design of compliant mechanisms using topology optimization. *Mechanics of Structures and Machines*, 25(4):493 – 524, 1997.

[96] O. Sigmund. A new class of extremal composites. *Journal of the Mechanics and Physics of Solids*, 48(2):397 – 428, 2000.

[97] O. Sigmund. Topology optimization: a tool for the tailoring of structures and materials. *Philosophical Transactions of the Royal Society A*, 358(1765):211 – 227, 2000.

[98] O. Sigmund. Manufacturing tolerant topology optimization. *Acta Mechanica Sinica*, 25(2):227 – 239, 2009.

[99] O. Sigmund and S. Torquato. Composites with extremal thermal expansion coefficients. *Applied Physics Letters*, 69(21):3203 – 3205, 1996.

[100] O. Sigmund and S. Torquato. Design of materials with extreme thermal expansion using a three-phase topology optimization method. *Journal of the Mechanics and Physics of Solids*, 45(6):1037 – 1067, 1997.

[101] O. Sigmund and S. Torquato. Design of smart composite materials using topology optimization. *Smart Materials and Structures*, 8(3):365 – 379, 1999.

[102] Troy R. Smith, Jeff Moehlis, and Philip Holmes. Low-dimensional modelling of turbulence using the proper orthogonal decomposition: A tutorial. *Nonlinear Dynamics*, 41:275–307, 2005.

[103] G. Stefanou and M. Papadrakakis. Stochastic finite element analysis of shells with combined random material and geometric properties. *Computer Methods in Applied Mechanics and Engineering*, 193(1-2):139 – 160, 2004.

[104] Raphael Sternfels and Phaedon-Stelios Koutsourelakis. Stochastic design and control in random heterogeneous materials. *International Journal for Multiscale Computational Engineering*, 9(4):425–443, 2011.

[105] V. Sundararaghavan and N. Zabaras. A multi-length scale sensitivity analysis for the control of texture-dependent properties in deformation processing. *International Journal of Plasticity*, 24(9):1581–1605, 2008.

[106] S. Torquato. *Random Heterogeneous Materials*. Springer-Verlag, 2002.

[107] S.S. Vel and A.J. Goupee. Multiscale thermoelastic analysis of random heterogeneous materials, part i: Microstructure characterization and homogenization of material properties. *Computational Materials Science*, 48(1):22 – 38, 2010.

[108] E. Wadbro and M. Berggren. Megapixel topology optimization on a graphics processing unit. *SIAM Review*, 4(51):707–721, 2009.

[109] Jingbo Wang and Nicholas Zabaras. A bayesian inference approach to the inverse heat conduction problem. *International Journal of Heat and Mass Transfer*, 47:2927–2941, 2004.

[110] X.F. Xu, X. Chen, and L.H. Shen. A green-function-based multiscale method for uncertainty quantification of finite body random heterogeneous materials. *Computers and Structures*, 87(21-22):1416 – 1426, 2009.

[111] C.L.Y. Yeong and S. Torquato. Reconstructing random media i and ii. *Physical Review E*, 58(1):224–233, 1998.

[112] N. Zabaras and B. Ganapathysubramanian. A scalable framework for the solution of stochastic inverse problems using a sparse grid collocation approach. *Journal of Computational Physics*, 227:4697–4735, 2008.